Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2017

Pedro José Neves Bispo

Análise quantitativa e qualitativa de controladores de redes definidas por software

A Software Defined Network Controller quantitative and qualitative analysis

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Pedro José Neves Bispo

Análise quantitativa e qualitativa de controladores de redes definidas por software

A Software Defined Network Controller quantitative and qualitative analysis

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Daniel Corujo, Professor adjunto convidado da Escola Superior de Tecnologia e Gestão de Águeda, e do Doutor Rui L. Aguiar, Professor catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor André Ventura da Cruz Marnoto Zúquete
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Doutor Daniel Nunes Corujo
Professor adjunto convidado da Escola Superior e Tecnológica de Águeda

Professor Doutor Paulo Manuel Martins Carvalho
Professor Associado da Universidade do Minho

agradecimentos /
acknowledgements

A presente dissertação significa o término de um ciclo que não seria possível sem a ajuda de várias entidades e pessoas, partilhando com elas o mérito que dela possa receber. À Universidade de Aveiro e Instituto de Telecomunicações agradeço a possibilidade de realização do presente trabalho.

Aos orientadores desta dissertação, Doutor Daniel Corujo e Doutor Rui L. Aguiar, manifesto o meu apreço por todos os conhecimentos transmitidos, colaboração, disponibilidade e ainda capacidade de estímulo demonstradas. Deixo também o meu agradecimento ao Flávio Meneses por toda a disponibilidade e ajuda demonstradas ao longo do desenvolvimento do presente trabalho.

Um agradecimento especial aos meus pais, Olívia e Bispo, e irmão, Renato pelo apoio incondicional e fundamental ao longo de todos estes anos. Agradeço ainda à restante família por todo o apoio demonstrado. Por último, agradeço a todas as pessoas que me acompanharam desde o primeiro dia na instituição e também aos colaboradores do projeto desenvolvido no âmbito da presente dissertação.

Palavras Chave

Redes 5G, Software-Defined Networking (SDN), OpenFlow, controladores SDN.

Resumo

Com o aumento do número de dispositivos ligados em rede, surgem novos desafios no ramo das redes. A necessidade de acompanhar o crescimento da utilização de dados móveis é um dos requisitos a ter em conta nas futuras redes 5G (5ª Geração), sobretudo em cenários de mobilidade. As redes controladas por software (do inglês, Software-Defined Networking (SDN)) permitem a simplificação e dinamismo necessários à criação das referidas redes 5G. As SDNs promovem ainda a separação do plano de controlo do plano de dados, permitindo um maior controlo, adaptabilidade e redução de custos.

O crescimento da tecnologia SDN levou ao desenvolvimento de diferentes controladores, com diferentes características. Existem vários controladores SDN, com origem em diferentes necessidades dos operadores e equipas de investigação. Este desenvolvimento individualizado tornou as comparações entre os controladores mais difíceis.

Deste modo, o trabalho desenvolvido fornece um estudo mais abrangente de vários controladores *open-source* (OpenDaylight (ODL), Open Network Operative System (ONOS), Ryu and POX), avaliando não só a sua performance como as suas características de uma forma qualitativa. Considerando a performance crucial nos controladores SDN, foram considerados vários critérios na avaliação dos controladores sob diferentes circunstâncias, utilizando a ferramenta *Cbench*. Os resultados apresentados são relativos à comparação qualitativa e quantitativa dos controladores em teste.

Keywords 5G networks, Software-Defined Networking (SDN), OpenFlow, SDN Controllers.

Abstract New challenges are being raised in the networking field with the increasing number of connected devices. The growth of mobile data usage has to be considered as a requirement for the deployment of future 5G networks, especially regarding mobility scenarios. Software-Defined Networking (SDN) enables a greater degree of dynamism and simplification for the deployment of those 5G networks. SDN provides the separation of the control plane from the forwarding plane, allowing more control, adaptability and cost reduction.
The growth of SDN integration in new mechanisms and network architectures led to the development of different controller solutions, with a wide variety of characteristics. Several SDN controllers exist, which originated from the different needs of operators and research teams. That resulted in the development of their own controller versions, which made comparison efforts more difficult. As such, this work provides a wider study of several open-source controllers, (namely, OpenDaylight (ODL), Open Network Operative System (ONOS), Ryu and POX), by evaluating not only their performance, but also their characteristics in a qualitative way. Taking performance as a critical issue among SDN controllers, several criteria were evaluated by benchmarking the controllers under different operational conditions, using the Cbench tool. Results are presented regarding both qualitative and quantitative comparisons between those SDN controllers under test.

Contents

List of Figures

List of Tables

Glossary

4G	4th generation mobile networks
5G	5th generation mobile networks
API	Application Programming Interface
BGP-LS	Border Gateway Protocol-Link State
CLI	Command-line Interface
DLUX	OpenDaylight User Experience
GUI	Graphic User Interface
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IRTF	Internet Research Task Force
ISP	Internet Service Provider
KVM	Kernel-based Virtual Machine
MUGV	Mobile User Generated Video
NB	Northbound
NETCONF	Network Configuration Protocol
NFV	Network Functions Virtualization
ODL	OpenDaylight
ONF	Open Network Foundation
ONOS	Open Network Operative System
OSGi	Open Services Gateway Initiative
OVS	Open vSwitch
OVSDB	Open vSwitch Database
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SAL	Service Adaptation Layer
SB	Southbound
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol
TL1	Transaction Language 1
TLS	Transport Layer Security
UHD	Ultra High-Definition
VM	Virtual Machine
VoD	Video on Demand
VSA	Video SDN Application
WMWG	Wireless & Mobile Working Group

CHAPTER 1

Introduction

Today we are witnessing the integration and proliferation of different kinds of Information and Communication Technologies into different fields of our society. The explosion of mobile access to on-line services, the Internet of Things and the consumption and production of rich real-time media content generate different kinds of data traffic, with widely different requirements and traversing disparate types of access networks and core infrastructure. Thus, the network of today is an assortment of complex mechanisms for managing and controlling these different aspects. Contributing towards this factor, lies the operational domains belonging to different stakeholders, who view the essence of the network in a different way: telco providers often focus on the operational and business perspective, while service providers aim for reachability optimization towards the users, and the users themselves want overall better service. Simultaneously satisfying involved actors is a highly complex task, whose harmonization is only achieved through careful planning and overprovisioning of networking resources.

In recent years, mobile data usage has been increasing continuously, and expected to keep evolving. Despite the increase of performance and capacity of mobile networks, future 5th generation mobile networks (5G) systems will need to cover mobility requirements, such as having multiple/heterogeneous radios access technologies (e.g. 5G and Wi-Fi)[1]. This increase in generated data and the need to dynamically adapt to changing situations in a cost-effective way, are demanding for more flexible and adaptive network control mechanisms. As a result, Software Defined Networking (SDN) has emerged.

SDN provides the separation of the network control plane from the forwarding plane, allowing more control, adaptability, agility and overall cost reduction. By having a complete view of the network, the SDN controller plays an extremely important role in such networks as it can manage the network structure and services dynamically.

Several SDN controllers exist, with most of them under continuous development. This diversity, which originated from the different needs of operators and research teams, resulted in the development of their own controller versions and made comparison efforts more difficult.

With the main goal of contributing with new results in this particular area, this work focuses on a key aspect of SDN-based 5G mobility management, namely the SDN Controller, and provides a wider study of several open-source controllers. Results highlighted a wide comparison of the most recent versions both quantitatively and regarding performance.

1.1 Motivation and Objectives

The growth of SDN integration in network architectures and new mechanisms, such as mobility, led to the development of different controller solutions, with a wide variety of characteristics. Despite existing studies, the most recent evaluations of SDN controllers are focused only on performance and are not up to date, since new versions of the most popular controllers are constantly being released. As such, this work provides a wider study of several open-source controllers, (namely, ODL, ONOS, Ryu and POX), by evaluating not only their performance, but also their characteristics in a qualitative way. Taking performance as a critical issue among SDN controllers, several criteria were evaluated by benchmarking the controllers under different operational conditions, using the *Cbench* tool. Based on this, the main objectives became:

1. Identification of the state of the art:
   Identifying the state of the art of SDN and its contribution to 5G mobility;
2. Study of SDN controllers and SDN open-source tools:
   An exhaustive study and exploration of SDN controllers and associated protocols and environments.
3. Elaboration of a comparative study of SDN controllers focusing their characteristics and performance:
   An extensive comparison between SDN controllers performing stress tests, as well as, a summary of their qualitative differences.

1.2 Methodology

The work began with a general study about mobile IP concepts like MIPv6[1], PMIPv6[2] and DMM[3], quickly shifting the focus into technologies related to 5G, namely SDN. After acquiring the necessary knowledge in SDN, the focus became mostly in SDN controllers.

Due to the number of SDN controllers under uninterrupted development to solve different needs from their developers, it became more difficult to compare them both qualitatively and quantitatively. That way, after identifying the lack of global comparisons among the most recent controllers, the work focused in a qualitative and quantitative study about SDN controllers.

[1]Mobility Support in IPv6,http://tools.ietf.org/html/rfc6275
[2]Proxy Mobile IPv6,http://tools.ietf.org/html/rfc5213
[3]Distributed Mobility Management,http://datatracker.ietf.org/wg/dmm/

Several metrics were used in a performance comparison between the elected controllers, as well as a qualitative analysis. Both results highlighted characteristics to take into consideration in a future choice of a SDN controller.

1.3 Contributions

This work explores SDN controllers and led to a qualitative and performance comparison between the elected controllers. Moreover, the study developed was accepted as a paper for International Young Engineers Forum on Electrical and Computer Engineering (YEF-ECE 2017), "A Qualitative and Quantitative assessment of SDN Controllers". Additionally, from a partnership with Universidade Federal de Uberlândia, a paper focused on the specifications of JAIN SLEE, "CREDENCE: A Carrier Grade Software Defined Networking Control Environment Based on the JAIN SLEE Component Model" was accepted to The 22nd IEEE Symposium on Computers and Communications (ISCC 2017).

This work was also presented at the 22nd Seminar of the *Rede Temática de comunicações móveis* (RTCM) and the International Young Engineers Forum on Electrical and Computer Engineering (YEF-ECE, 2017) conference.

Also, within the industry, this work was also integrated into a collaborative project with Altran, named "VDSNET-Enhanced control over mobile upload video", shown at the students@deti event 2017.

1.4 Master Thesis Layout

The layout of the master thesis is organized as follows: Chapter 2 presents a state of the art about SDN, followed by Chapter 3 where the specifications of both comparisons are detailed. The setup is evaluated in Chapter 4, as well as the qualitative comparison, presenting the results. Finally, the master thesis concludes in Chapter 5, pointing out future work as well.

CHAPTER 2

State Of The Art

SDN enables a greater degree of dynamism and simplification for the deployment of future 5G networks [2]. With the increasing number of connected devices, new challenges are being raised in the networking field. Also, a recent forecast [3] pointed out that global IP traffic reached 1.1 zettabytes per year by the end of 2016 and will reach 2.2 zettabytes by 2020. From that amount of data, IP video traffic will be 82 percent of all IP traffic (both business and consumer) by 2020, up from 70 percent in 2015. Additionally, consumer Video on Demand (VoD) traffic will nearly double by 2020. Not only video traffic is increasing quickly, but also its quality is following up, by 2020, more than 40 Percent of connected Flat-Panel TV Sets will be 4K. This increasing expectation for high video quality has an impact on networks, on user engagement [4] exploring how video quality affects user experience. Additionally, a recent study [5] explores user's Quality of Experience (QoE) expectations taking into consideration low-latency and Ultra High-Definition (UHD) video, knowing that UHD streams are predicted to require up to 16 times as much bandwidth as current High-Definition streams.

Targeting technologies, in 2021, 4th generation mobile networks (4G) traffic is expected to represent 53% of mobile connections, compared to only 26% in 2016. Despite that, 4G traffic accounted for 69% of mobile traffic in 2016, since, on average, a 4G connection represented four times more traffic than a 3G connection [6]. This general growth, let to the fastest development of 5G related technologies, within which, SDN.

When considering both SDN and Network Functions Virtualization (NFV), their market is foreseen to be worth hundreds of millions by 2020 [7]. Three-quarters of that value is predicted to come from big operators such as AT&T, Verizon Communications, Orange, Telefónica, among others.

Wireless and mobile devices will account for 63 percent of IP traffic in 2021, and wired devices will account for 37 percent of IP traffic [3]. In 2016, wired devices accounted for the majority of IP traffic, at 51 percent. According to the same authors, smartphone traffic will exceed PC traffic, since in 2016, PCs accounted for 46 percentage of total IP traffic against 13 percent for smartphones. In 2021, PCs will account for only a quarter of IP traffic while

smartphones are foreseen to account for a third of total IP traffic. Also, of all IP traffic (fixed and mobile) in 2021, 50% will be Wi-fi, 30% will be wired, and 20% will be mobile[6].

Decoupling the control plane from the data plane brought a new set of possibilities. In such networks, the controller plays a major role by being able to manage forwarding entities, such as switches, through the application of flow-based rules via a southbound interface. In turn, the controller itself can be managed by means of actions and policies provided by high-level network functions, via a northbound interface.

This work will focus on a key enabling cornerstone technology of upcoming 5G networks mobility, namely SDN.

2.1 Software-Defined Networking

The need to manage enormous amount of data and therefore complex computer networks has resulted into new business models. Network architectures in which the control plane is decoupled from the data plane have been growing in popularity. SDN brings promising opportunities to network management in terms of simplicity, programability and elasticity (beneficting mechanisms, such as mobility). Among the main arguments for SDN is that it provides a more structured software environment in order to develop network-wide abstractions [8].

SDN is based on three architectural principles, as follows [9]:

1. **Separation of the network control plane from the forwarding plane:**
   The main goal of this principle is to allow independent deployment of control and traffic forwarding and processing entities. It is a necessary precondition of centralized control. In addition, this separation also allows for different optimization of platform technology and software life cycles. Within the architecture, decoupling originates an entity called SDN controller, which has a management-control responsibility.
2. **Logically centralized control:**
   Like stated in the previous topic, decoupling of traffic forwarding and processing from control is a prerequisite of centralized control. Nonetheless, it is important to acknowledge that being logically centralized means being controlled by a single entity, even if that implementation is in a distributed form.

   In a centralized control principle, within a wider perspective, a SDN controller can use resources more efficiently. As in any other principle, there are disadvantages that must be highlighted, such as:
   - the increased complexity alongside scalability;
   - by being centralized, it becomes an easier target regarding the security of the network. Management-control information exchange is highly constrained across trust boundaries;
   - the necessity of co-existence of SDN and non-SDN technologies.

3. **Programmability of network services:**
   Allowing a client to exchange information with an SDN controller is the main objective of this principle. It may happen during the lifetime of a service or even prior to the establishment of that service according to changes in client needs or the state of the client's virtual resources. Agility is the main outcome when allied to the ability of SDN to dynamically create new resources on demand or modifying the existing resources, especially virtual network functions.

This technology seems to have appeared suddenly, but SDN is part of a long history of efforts to make computer networks more programmable [10]. As one of the most significant paradigm shifts recently seen in the networking industry, most of the principles behind SDN, such as, network programmability has begun since the 1990s and even the separation of control plane was proposed in the 2000s by Internet Engineering Task Force (IETF) [11]. Attached to this progress is the evolution of the OpenFlow specification process at Stanford University prior to the creation of Open Network Foundation (ONF).

SDN is also consolidated by the concept of Internet of Things (IoT), following the exponential growth of devices that exchange information over the network, revealing the inefficiency and rigidity of traditional architectures [12]. Additionally, advantages mentioned above make SDN a suitable paradigm for Internet Service Providers (ISPs) to share Quality of Service (QoS) control with external entities.

2.1.1 SDN Arquitecture

The architecture, which describes principles, components and roles in abstract ways, is supposed to create a common understanding that facilitates parallel development. Having a common information model leads to greater consistency since different implementations can be compared amongst themselves.

Considering an architecture like an incomplete collection of perspectives under certain ideas, it is supposed to facilitate the development of concepts into realities. Regarding how recent SDN technology is, and how much it still has to develop, an open architecture reduces the difficulty of extension in previously unseen directions.

The main objective of such an architecture is to help providers in better serving their customers, mainly by reducing their own cost to deliver those services [9]:

- Reducing the time and cost of developing new services;
- Flexibility regarding the definition and availability of resources, including virtual network functions (VNFs);
- Efficiency on resource loading, with continuing real-time optimization, facilitating global agreement with a common information model;
- Assembly of resources into services, as well as, merging traditional business and operations support system functions with control.

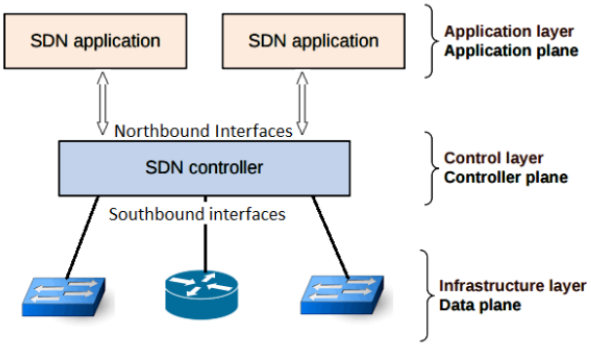


Figure 2.1: SDN architecture [13]

Based on Figure 2.1, the three basic components can be summed up as: the controller plane, the data plane and the management plane. Communication is made by their interfaces: a Northbound (NB) interface for communication between the application and the controller plane and a Southbound (SB) interface between the controller and the infrastructure layer. Unlike the SB interface, the NB interface is still undefined, despite existing efforts to reach a standardization [14].

2.1.1.1 Data Plane

SDN decoupled the data and control planes in order to develop the network into a more programmable and flexible state. The data plane (sometimes known as the user plane, forwarding plane or carrier plane) is the part of a network that processes user traffic, by being responsible for carrying the user traffic according to the control plane logic.

This plane includes the network elements, allowing the communication between data and control planes through protocols, such as the OpenFlow protocol. This way, the traffic can be monitored and managed without having to manually reconfigure individual switches.

2.1.1.2 Control Plane

In SDN, the network intelligence is logically centralized in software-based controllers. The control plane performs different functions like routing, traffic engineering, security and mobility, by having a global network view and by being able to set the configuration of each network device. In general, functions of the control plane include system configuration and management. The network OS is also included in the control plane, in other words, SDN controller.

2.1.1.3 SouthBound Interface

The link between the data plane and the control plane represents an important aspect of SDNs. Since these forwarding elements are managed by an open interface, through the SB interface, it becomes more important that this link remains available and secure [15]. There are several protocols to implement these controller-switch interactions, defining the communication between the switching hardware and a network controller, such as the OpenFlow protocol, NETCONF, etc, that will be discussed further in Section 2.1.3.

2.1.1.4 NorthBound Interface

Allows the controller to be interfaced by high-level entities. For example, external management systems or network services may wish to extract information about the network or even control an aspect of network behaviour, such as, detecting optimal mobility opportunities[16][17]. Unlike SB communications, there is no currently accepted standard for NB interactions.

2.1.2 SDN Controllers

The SDN controller satisfies client requests by virtualizing and orchestrating its underlying resources. As the network environment changes and client demands change as well, the SDN controller is responsible for continuously updating network and service states towards a policy-based optimum configuration.

As mentioned before, the growing number of SDN controllers kept under development from distinct natures increased the difficulty of ranking those controllers under any requirement. The evaluated controllers election is described further in section 3.1.1 and the following subsections will describe them.

2.1.2.1 OpenDaylight

OpenDaylight[1] is an open-source project maintained as part of the Linux foundation, founded in 2013. Of all members contributing to the project, the highlight goes to companies like IBM, Cisco, Juniper Networks, VMware, NEC, Microsoft, Ericsson and several other major networking vendors. To sum up, it is widely supported by industry members and researchers.

ODL's architecture allows users to control applications, protocols and plugins, allowing optimization of the networks to current needs but also an easier adaptation to any changing requirements. OpenDaylight's platform can be configured in any number of ways since the modularity and flexibility of ODL allows end users to select whichever features matter the most to them, creating controllers that meet their needs.

ODL is written in Java, with the main decisions voted by an elected committee[2]. Up until now, six releases have been launched (Hydrogen (Feb-2014), Helium (Oct-2014), Lithium (Jun-2015), Beryllium (Feb-2016), Boron (Nov-2016) and Carbon (May-2017). Figure 2.2 shows the architecture of ODL Beryllium[18], released in early 2016.

[1] http://www.opendaylight.org/
[2] https://www.opendaylight.org/governance

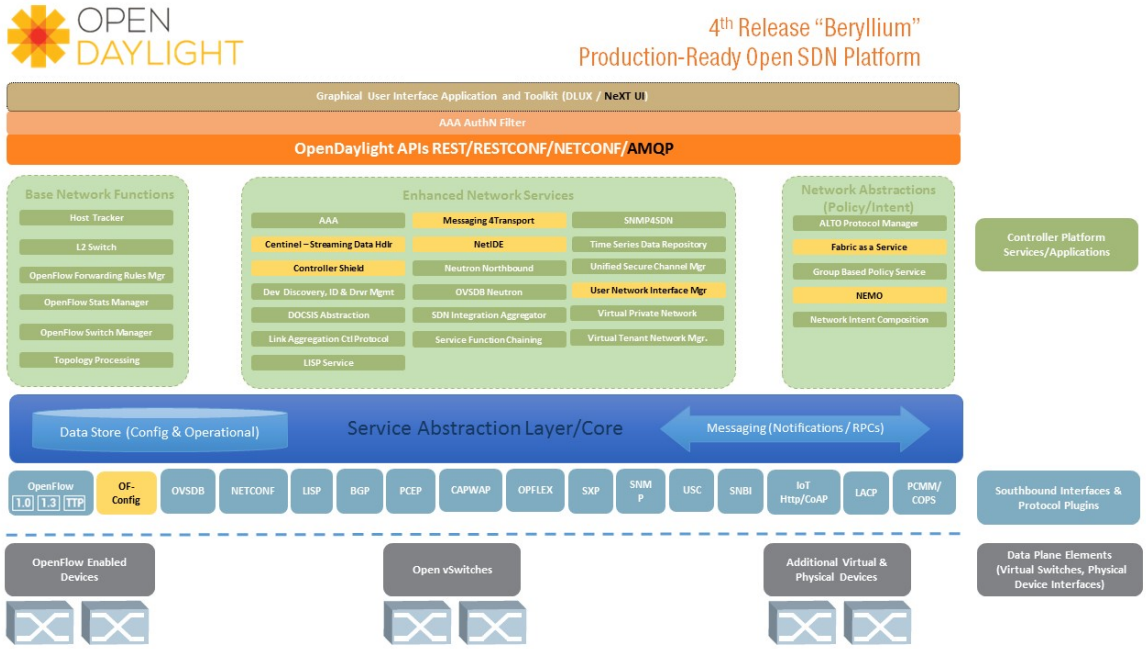


Figure 2.2: OpenDaylight SDN controller architecture (Beryllium) [18]

ODL SDN Controller has several layers that can be easily separated into three main layers (top, middle and bottom). The top layer includes network applications and services, a controller as the middle layer that manages network devices and the bottom layer that consists of physical and virtual devices.

In one hand, ODL exposes an open northbound Application Programming Interface (API) that supports Open Services Gateway Initiative (OSGi) framework or bidirectional Representational State Transfer (REST), whether the applications will run in the same address space as the controller or not. These applications above the middle layer may use the controller to gather information, perform analytics or run algorithms and then implement new rules throughout the network using the controller.

On the other hand, the southbound interface supports multiple protocols, such as, OpenFlow (multiple versions), Open vSwitch Database (OVSDB), NETCONF, Transaction Language 1 (TL1), Border Gateway Protocol-Link State (BGP-LS), Simple Network Management Protocol (SNMP), among others, as separate plugins. These modules are linked dynamically into a Service Adaptation Layer (SAL) that figures out how to implement the requested service no matter which protocol is being used between the controller and the network devices.

2.1.2.2 Open Networking Operative System

ONOS[3] is an open-source SDN controller project driven by ON.Lab[4], a non-profit organization founded in 2012 with the assistance of Stanford University. As members contributing for the project, the highlight goes to companies like Cisco, AT&T, Google, Huawei, SKTelecom,

[3]http://onosproject.org/
[4]OpenNetworks Laboratory

Intel, Verizon and several other major networking vendors. ONOS is positioned with a focus on scalability and performance for telecommunication companies and service providers. ONOS is a Java based platform. The latest release, "Kingfisher"(1.10) in June 2017, was the tenth main release since the end of 2014, launching an average of four new versions every year.

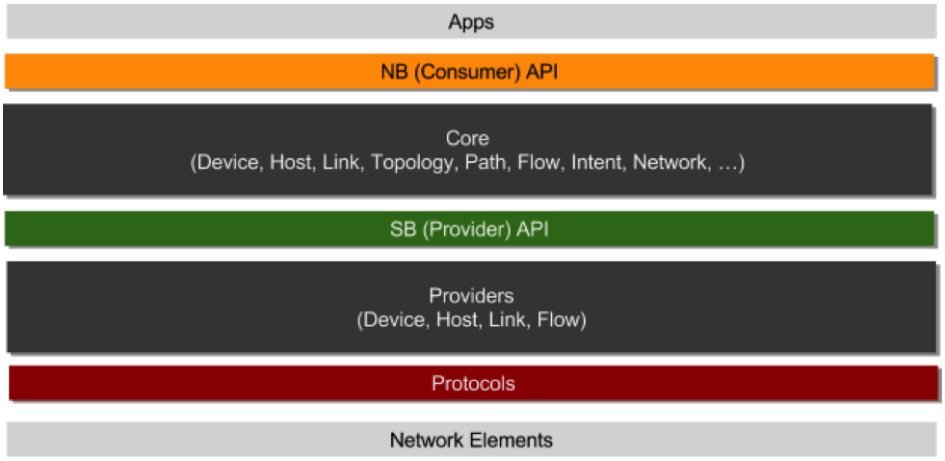


Figure 2.3: ONOS architecture [19]

ONOS is architected with tiers of functionality[19], as shown in Figure 2.3. Regarding interfaces, ONOS provides a flexible and extensible API with multiple layers of abstraction for both network programming and configuration. Regarding northbound interfaces, ONOS provides two: the global network topology view, which can be used to calculate paths, provision flows and perform other network functions and the intent framework to calculate the best path between two points. Otherwise, the southbound interface supports multiple protocols, such as, OpenFlow (multiple versions), OVSDB, NETCONF, SNMP, TL1, among others. A REST API, a Command-line Interface (CLI) and a Graphic User Interface (GUI) are available interfaces to administer networks using ONOS.

2.1.2.3 Ryu

Ryu[5] is a SDN controller supported by NTT[6] that provides a well-defined API to create new network management and control applications. It has been certified to work with Open vSwitch (OVS) and offerings from Hewlett Packard, IBM and NEC.

[5] https://osrg.github.io/ryu/
[6] Nippon Telegraph and Telephone Corporation

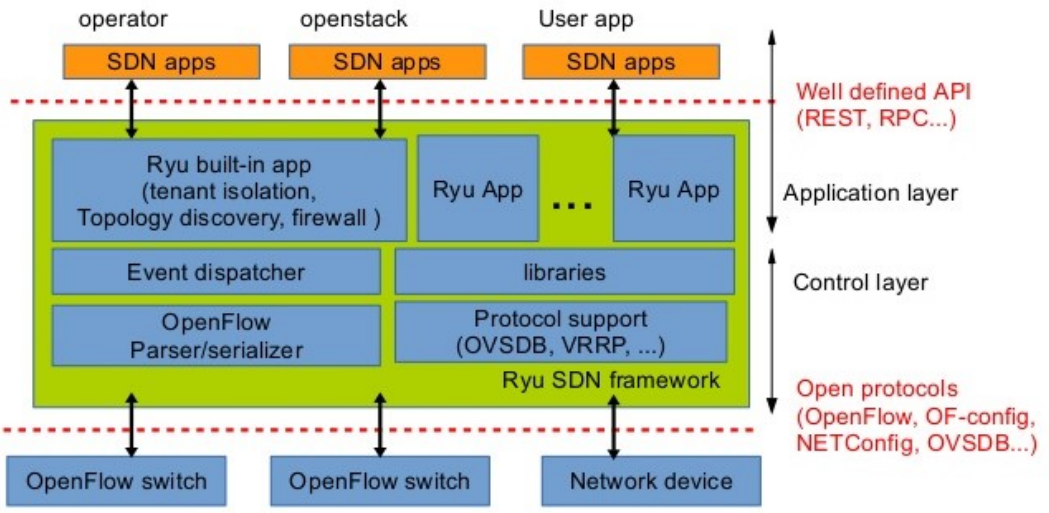


Figure 2.4: Ryu architecture

Written in Python, the southbound interface supports multiple protocols, such as, OpenFlow (multiple versions), OVSDB, NETCONF, SNMP, among others.

2.1.2.4 POX

POX[7] inherited from the NOX controller (Nicira). It is also an enabling framework to interact with OpenFlow switches. POX supports OpenFlow and OVSDB and it is a popular tool for teaching and researching. POX claims the following advantages over NOX [20]:

- POX has a Pythonic OpenFlow interface.
- POX has reusable sample components for path selection, topology discovery, and so on.
- POX specifically targets Linux, Mac OS, and Windows.
- POX supports the same GUI and visualization tools as NOX.
- POX performs well compared to NOX applications written in Python.

2.1.2.5 Other SDN Controllers

Knowing how many SDN controllers exist, most still being updated, the choice of those elected was made based on a growing list.

NOX[8][21] is called the original OpenFlow controller, initially developed by Nicira Networks in 2009. NOX provides a C++ API to OpenFlow and an asynchronous, event-based model. Also a notation for NOX-MT, a slightly modified multithreaded successor of NOX [8].

Beacon[9] is a Java-based open source OpenFlow controller created in 2010 and is the basis of Floodlight. Beacon is multithreaded and has the capability to add and remove applications (runtime modularity), without shutting down the Beacon process [22].

[7]https://github.com/noxrepo/pox
[8]https://github.com/noxrepo/nox
[9]https://openflow.stanford.edu/display/Beacon/Home

Floodlight[10] is a Java-based SDN Controller offered by Big Switch Networks. Floodlight is built on work that began at Stanford University and works with the OpenFlow protocol to orchestrate traffic flows, presenting REST APIs that make it easier to program.

OpenMul[11] has a C language based multi-threaded infrastructure at its core. It supports a multi-level northbound interface for hosting applications and can control network devices supporting OpenFlow, OVSDB, as well as NETCONF.

Trema[12] is an OpenFlow controller programming framework that provides everything needed to create OpenFlow controllers in Ruby and C, created by NEC.

Maestro is a multithread controller developed in Java [23].

2.1.2.6 Controllers Comparison

Despite existing studies, the most recent evaluations of SDN controllers are focused only on performance and are not up to date, since new versions of the most popular controllers are constantly being released. Existing literature is mostly focused on performance [24] [25] [26] [27] and few studies have focused on the comparison of qualitative aspects [28], but did not cover these elected SDN controllers under evaluation.

2.1.3 SDN Open-source tools

In this subsection some of the most relevant protocols will be discussed.

2.1.3.1 OpenFlow Protocol

One of the most used SDN protocols is OpenFlow. An OpenFlow-enabled switch, also known as "OpenFlow Switch" is characterized by a flow table (may contain one or more flow tables), a secured channel that realises the connection with the controller and the OF protocol as the way to communicate with the controller.

The original concept begun at Stanford University in 2009. Regarding the OF protocol evolution (Fig.2.5), the first version of OpenFlow protocol was released in 2009 and it was limited to a single flow table with three components in a flow entry, namely Header Fields, Counters and Actions, with the header field containing only 12 fixed match fields. Due to these limitations, version 1.1 emerged, which introduced multiple tables and a group table, encompassing group entries. Despite those enhancements, the use of a fixed length structure for match statements was limiting the flexibility. In that way, OF 1.2 introduced Type-Length-Value (TLV), which allowed new match fields to be added in a modular way, increasing their flexibility. Also, version 1.2 added basic support for IPv6.

[10] http://www.projectfloodlight.org/floodlight/
[11] http://www.openmul.org/
[12] https://trema.github.io/trema/

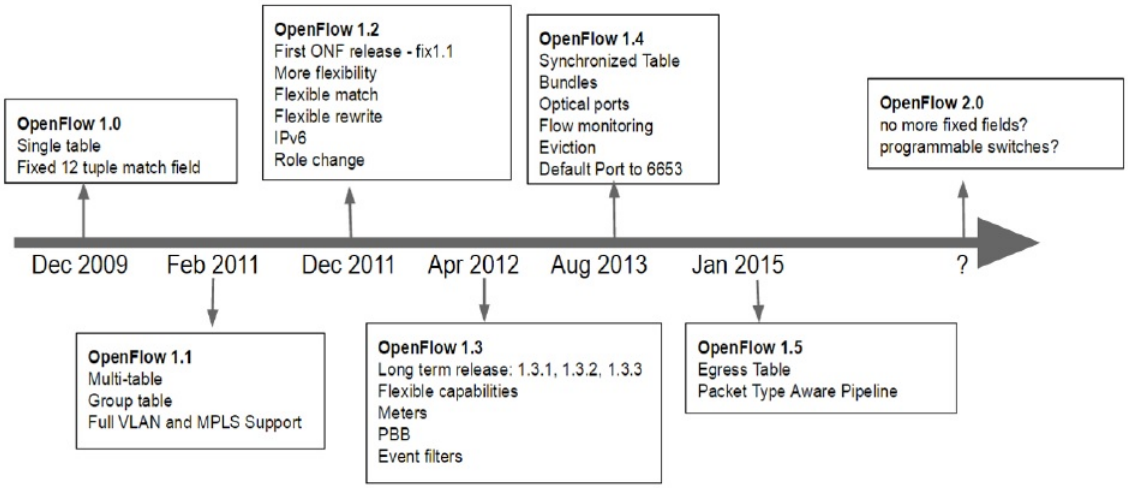


Figure 2.5: OpenFlow timeline [29]

Version 1.3, released in 2012, brought two main additions to the OpenFlow protocol. The flow table was extended with a "tablemiss" entry, allowing the processing of non-matched packets to be more flexible, since in previous versions it would be dropped or sent to the controller in a *packet-in* message. In addition, considering QoS, OpenFlow 1.3 introduced a "Meter table", which extended QoS capabilities into OpenFlow. Also, version 1.3 provides optional support for encrypted Transport Layer Security (TLS) communication and a certificate exchange between the switches and the controller. OpenFlow 1.4 continues increasing scalability by introducing a new "Synchronized table", where flow tables can by synchronized unidirectionally or bidirectionally. Also, "Bundles" were introduced, which grouped state modifications together into a transactional group. "Scheduled Bundles" were introduced in version 1.5, allowing synchronization among multiple switches.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flag

Table 2.1: Components of a flow entry in OpenFlow 1.5.

The last version of OpenFlow available also introduced the "Egress table", adding the possibility to match packets based on its output port. Currently, vendors are focusing in providing OpenFlow versions 1.0 and 1.3 in their products. Very recently, in 2016, Pica8 claimed to be the first manufacturer to add support for OpenFlow 1.5 in their products, having also provided support for OpenFlow 1.4 in 2014. OpenFlow, as a protocol that originally focused on interactions between a controller node and network switches, was recently extended in experimental research studies to terminal nodes [16] [17] allowing direct SDN interactions between the network and the mobile node. In this way, mobility processes are able to become optimized, allowing the mobile node to provide information about connectivity targets and conditions, using a single control protocol. By extending SDN mechanisms to the mobile nodes, the controller gains the ability to manage data flows all the way to the terminal node.

2.1.3.2 OVSDB

OVS was created by the team at Nicira, that was later acquired by VMware. OVS was intended to meet the needs of the open-source community, since there was not a feature-rich virtual switch offering designed for Linux-based hypervisors, such as Kernel-based Virtual Machine (KVM) and XEN. In the same way that software-based SDN controllers can co-exist with hardware based solutions, OVS allows existing supportive switching devices to incorporate SDN features. The need to manage these implementations gave magnitude to OVSDB[30]. The management interface of OVSDB is used to manage and configure operations on the OVS instance. Among all the operations that are supported by OVSDB we can highlight the ability to create, modify and delete OpenFlow datapaths, as well as, ports and tunnel interfaces on OpenFlow datapaths. Also, it supports the configuration of QoS policies and attachment of those policies to queues.

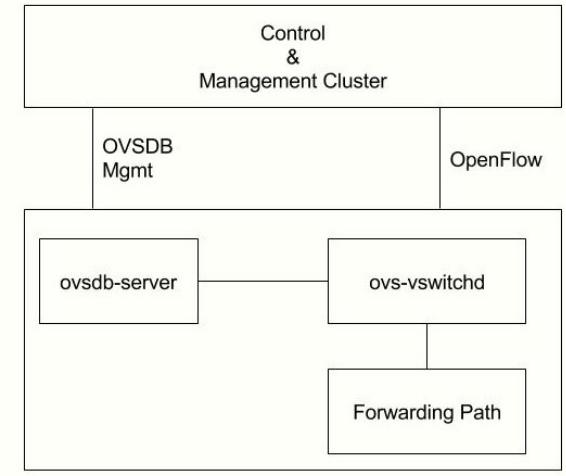


Figure 2.6: Open vSwitch Interface

2.1.3.3 NETCONF

The NETCONF protocol [31] was developed by the IETF and first published in 2006 and is defined as a simple mechanism through which a network device can be managed. In other words it provides mechanisms to install, manipulate and delete the configuration of network devices. As the protocols discussed above, this protocol allows the device to expose an API that can be used to send and receive configuration data sets.

NETCONF can be partitioned into four layers according to Figure 2.7:

- Content layer: includes the content and notification data;
- Operations layer: defines a set of base protocol operations invoked as Remote Procedure Call (RPC) methods;
- Messages layer: provides a mechanism for encoding RPCs and notifications;
- Secure Transport layer: provides a communication path between the application and the network device.

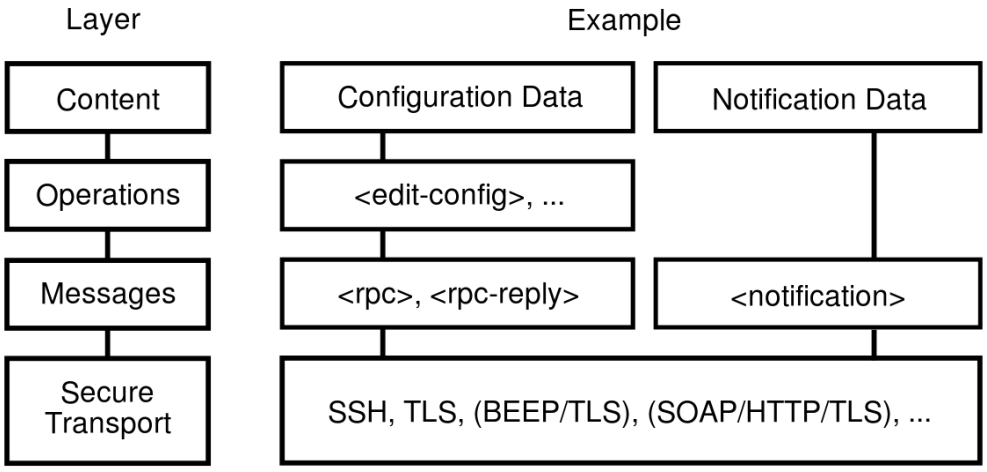


Figure 2.7: NETCONF Protocol Layers [31]

2.1.4 Standardization

The concept of SDN has attracted recently attention from network carriers and service providers because of its potential to provide flexible/dynamic control and programmability in network topology and packet forwarding/processing functions. ONF[13] is an organization that allies the promotion and adoption of SDN with the design of the OpenFlow protocol. They also focus on standardizing both northbound and southbound interfaces. The Wireless & Mobile Working Group (WMWG) collects use cases and determines architectural and protocol requirements for extending ONF based technologies to wireless and mobile domains. Also, as part of the Institute of Electrical and Electronics Engineers (IEEE), the Open Mobile Network Interface for Omni-Range Area Networks (OmniRAN) supports the use of SDN principles, lowering the barriers to meet network technologies, operators and service provides. Unlike controller-switch communications, there is no currently accepted standard for northbound interactions and they are more likely to be implemented on an ad hoc basis for particular applications [15].

2.1.4.1 IETF

The IETF[14] is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

The IETF's Forwarding and Control Element Separation (ForCES) Working Group has been working on standardizing mechanisms, interfaces, and protocols aiming at the centralization of network control and abstraction of network infrastructure. The Internet Research Task Force (IRTF)[15] promotes the research of the Internet evolution by creating a

[13]https://www.opennetworking.org/
[14]https://www.ietf.org
[15]https://irtf.org/

long-term research group. IRTF is also focusing on SDN under various perspectives with the goal of identifying new approaches that can be defined and deployed, as well as identifying future research challenges.

2.2 SDN-enabled Environments

This section presents the most relevant solutions used in this work for emulation and simulation of SDN networks.

2.2.1 Mininet

Mininet [16] is defined as a network emulator for SDN systems that creates virtual hosts, switches, controllers and links. It runs this collection on a single Linux kernel.

The main features to highlight are the following:

- a command-line launcher to instantiate networks;
- a handy Python API for creating networks of varying sizes and topologies;
- full API documentation;
- parametrized topologies;
- a CLI which provides useful diagnostic commands like iperf and ping;
- a "cleanup" command to get rid of interfaces, processes, etc.

Open vSwitch is used as the default OpenFlow switch in Mininet, enabling a fast framework deployment. Despite existing studies regarding mininet performance [32], it is possible to sum its characteristics up in a general way. In one hand, Mininet greatest characteristics are:

- it is fast, it is possible to start up a simple network in just a few seconds;
- ability to create custom topologies, from a single switch to larger Internet-like topologies, a data center, or anything else;
- possibility to run real programs, basically, everything that runs on Linux (Wireshark as an example);
- customization packet forwarding: Mininet's switches are programmable using the OpenFlow protocol.
- Mininet can run on a laptop, on a server, in a Virtual Machine (VM) or in the cloud;
- ability to share and replicate results;
- interaction using Python scripts;
- it is an open source project that is under active development.

On the other hand, despite running on a single system it imposes resource limits. Mininet uses a single Linux kernel for all virtual hosts, meaning this software can not run on Windows or other operating systems kernels. To achieve custom routing or switching behaviour, there is the need to use a controller with the required features.

[16] http://mininet.org/overview/

2.2.2 OpenStack

OpenStack [17] in an open source software for building public, private and hybrid clouds. Originally developed by NASA and Rackspace, OpenStack consists of three core software projects. Project *Nova* refers to OpenStack Compute Infrastructure, *Swift* to OpenStack Object Storage Infrastructure and *Glance* to OpenStack Image Service Infrastructure[33]. OpenStack Icehouse, released in 2014 was the software version used in this work.

2.2.3 Proxmox

Proxmox[18] is an open source software that can manage VMs and clusters based on KVM and OpenVZ. Based on debian, Proxmox offers full virtualization and paravirtualization with the support of a host OS [34].

2.3 Evaluation Tools

This section presents the description of *Cbench*[19], the evaluation tool used for the performance tests. Despite the existence of an extended version of *Cbench* (*MT-Cbench*[20] that supports multithreading), HCProbe[21] [35] or OFCBenchmark tool [26], this section will only cover *Cbench*.

2.3.1 Cbench

Cbench [22] is an often-used tool for SDN controller benchmarking. It interacts with the controller in latency or throughput mode. In latency mode, *Cbench* sends a *packet_in* (Figure 2.8) to the controller and waits for the *flow_mod* (Figure 2.9) to come back, measuring how many times this process occurs in a second. On the other hand, in throughput mode, *Cbench* queues *packet_ins* and counts *flow_mods* as they come back. *Cbench* has a lot of input arguments:

- ip address and Port where the controller is running;
- number of milliseconds per test;
- amount of loops per test;
- number of switches to emulate;
- number of hosts to emulate per switch;
- starting test delay;
- warmup - number of ignored loops;
- cooldown between loops;

[17] https://www.openstack.org/
[18] https://www.proxmox.com/
[19] https://sourceforge.net/projects/cbench/
[20] https://github.com/intracom-telecom-sdn/mtcbench
[21] https://github.com/ARCCN/hcprobe
[22] https://sourceforge.net/projects/cbench/

- running mode (latency or throughput).

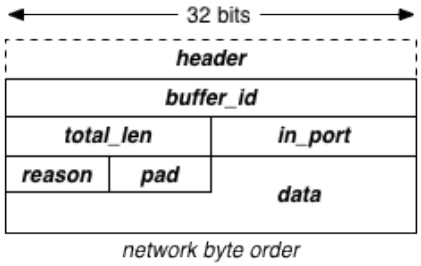


Figure 2.8: *Packet_in* network byte order

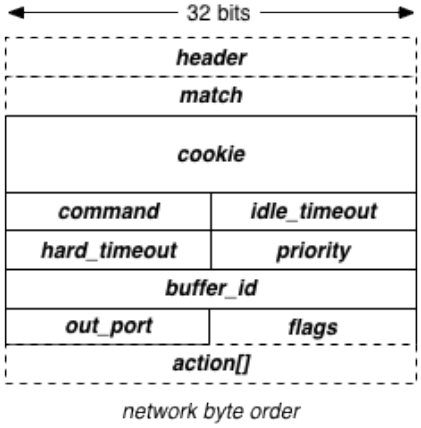


Figure 2.9: *Flow_mod* network byte order

2.4 Chapter Considerations

This chapter explored the state of the art of SDN, one of the key cornerstones for the enablement of 5G and mobility scenarios therein. Besides the SDN architecture, where its layers and interfaces were explained, the available SDN controllers were also studied, emphasizing the four elected for this work. In addiction, some controllers comparisons were also presented. The most used protocols were described as well as SDN-enabled environments that were studied and slightly compared. Last, the benchmarking tools were presented.

CHAPTER 3

Design and Implementation

This chapter will explore the characteristics taken into consideration regarding the qualitative comparison, as well as, the elected controllers. The selected criteria to analyse each controller are presented into detail.

Also, it presents the setup requirements and detailed information about *Cbench* and its modes of operation, latency and throughput mode. Last, the framework deployment is presented.

Last, the VDSNet project is presented with its high level operation.

3.1 Introduction

In order to compare SDN controllers, several criteria had to be picked from an endless list of possible technical features. After electing the SDN controllers to analyse, each relevant characteristic was compared on each controller.

Further, after analysing the *Cbench* tool in detail, diversified arguments were pointed out in order to achieve latency and performance results, by running *Cbench* in latency or throughput mode, respectively. Each test was run under the same conditions, whenever possible.

Two types of OpenFlow messages are crucial for the tests carried out:

- Packet_in:
  The packet_in is an asynchronous message sent by the switch to the controller without its solicitation. The asynchronous messages are the ones initialized by the switch to send any update to the controller.
- Flow_mod:
  The "modify flow entry message" (i.e., flow_mod) is a modify-state message, that belonging to the Controller-to-switch messages group, is sent by the controller to manage switches state.

Either running modes involve the exchange of these type of messages as explored into detail further in this chapter.

3.1.1 Evaluated Controllers

From a growing list of popular SDN controllers, the option went for two Python-based controllers (i.e., POX and Ryu) and two Java-based controllers (i.e., ODL and ONOS). At the same time, it was intended to have recognized controllers from both research and business worlds.

All controllers are open-source. The majority of developed comparisons were based on the following controllers versions: ONOS version was Hummingbird(1.7.0), ODL was Beryllium(SR3) and Ryu was version 4.7. Presently, POX has lost official support and the last version available was considered. Occasionally, a different version of a specific controller is tested and compared to another version of the same controller.

3.2 Qualitative Comparison

In this section the qualitative comparison process will be discussed by pointing out the selected criteria to compare. These criteria are also described under detail for a better understanding of the impact they might have.

3.2.1 Evaluation Criteria

To qualitatively compare the controllers, a set of criteria was selected to analyse each controller individually, such as, their interfaces, programming language, modularity and multi-threading support, as well as their documentation.

As the communication with higher and lower components of the network is made through their NB and SB interfaces, respectively, both components were analysed as well as the support of SB protocols such as OpenFlow, OVSDB, NETCONF, among others. Summarizing the functionality of NB interfaces, ODL presents two interfaces for applications: a web-based REST API interface and the OSGi interface. Additionally, multiple interfaces are presented in web-based interface OpenDaylight User Experience (DLUX)[36], providing access to the controller's functions as visualized REST and YANG interfaces. There is the ODL YANG UI [37] module that is a collection of all accessible REST API's in the controller with some information about data structures. A schematic of interconnections of devices connected to the controller is provided by the ODL topology module.

On the other hand, ONOS presents an Intent Framework and the global network topology view as northbound interfaces. Additionally, a web-based GUI, a REST API and a CLI. POX brought some limitations regarding its northbound interface from its sibling NOX. Ryu

supports REST and uses WSGI[1] to create the REST APIs. The possibility of using a GUI and the support of REST API are the highlights taken into account.

The development language used and the versions of OpenFlow supported were also considered mainly in a development scenario.

Regarding the modularity of each controller, in Ryu or POX there is the need to stop the controller and run it again with the needed modules for the REST methods to execute, while ODL and ONOS provide the ability to add several functionalities on the go.

Documentation available[2][3][4][5] for each controller was analysed and compared to their competitors. There are two main types of documentation that were considered useful to analyse the controllers. Every controller presents official documentation regarding their controllers and with the exception of POX, they also present information regarding most of controllers' features and modules. Additionally, every controller under study presented several tutorials and use-cases in an unofficial way, from Github[6] to personal pages or forums.

Further, multi-threading support was also taken into consideration in the qualitative comparison, despite impacting also on the performance comparison.

3.3 Performance Comparison

In this section, a detailed analysis of the performance tests is provided. With the setup requirements presented first, followed by the benchmarking tool *Cbench* and then the framework deployment.

3.3.1 Setup requirements

Initial tryouts run in two virtual machines under OpenStack, running Ubuntu 16 server distribution with 2 cores and 2 GB of RAM each. Hardware requirements are difficult to define, since each controller has different needs and it also depends on factors such as the managed network size or the number of messages exchanged with network devices. It was quickly noticed that they were especially demanding of processing power mainly in the machine hosting the SDN controller, since ODL and ONOS require Karaf[7] that needs for itself more than 2GB of RAM at certain times. After being upgraded and understanding that different controllers need distinct computing power, a reasonable setup was prepared with the following characteristics for both machines:

[1] Web Server Gateway Interface
[2] http://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/release_notes.html
[3] https://wiki.onosproject.org/display/ONOS/Tutorials
[4] https://github.com/noxrepo/pox
[5] https://ryu.readthedocs.io/en/latest/
[6] https://github.com/
[7] http://karaf.apache.org/

	Controller	Cbench
Operating System	Ubuntu 16.04_server	Ubuntu 16.04_server
CPUs (cores @ 3GHz)	4	4
Memory (Gb)	4	4

Table 3.1: Test scenario characteristics for performance tests

Interactions between machines are explained further in section 3.1. From previous tests, it was known that available computer power may influence test results, mostly when considering power-hungry SDN controllers as ODL or ONOS. Also, only the fundamental features were installed in ODL, since each additional feature increases the computer power needed.

3.3.2 Cbench tool

The benchmarking tool used was *Cbench* and since it has two working modes, those will be discussed in further subsections. *Cbench* interacts with the controller by sending a *packet_in* and receiving a *flow_modification* in different ways, as described in the following subsections. Despite the different characteristics, the executed tests focused on providing the same running environment to each controller, with *Cbench* being configured with the same parameters whenever possible. Of all parameters this tool handles, the number of switches emulated and the number of MACs/hosts to emulate per switch were often changed in order to perform several scalability tests, with the results on the different configurations. The minimum time for each test, as well as, the addition of a warmup delay were added accordingly to each controller need.

3.3.2.1 Latency Mode

In latency mode, *Cbench* sends a *packet_in* to the controller and waits for the matching flow to come back, measuring how many times this process occurs in a second. The latency result is the elapsed time between sending the *packet_in* and receiving the *flow_mod*. This mode is usually used to compare the SDN controller's latency by emulating just one switch, but it can be also used to verify the latency impact of increasing the number of emulated switches and also varying the number of hosts emulated per switch.

3.3.2.2 Throughtput Mode

In throughput mode, *Cbench* queues *packet_ins* and counts *flow_mods* as they come back. A lot of different tests can be run using *Cbench* in throughput mode in order to achieve a performance comparison between SDN controllers. The most impactful tests are: throughput switch scalability and throughput with an increasing number of hosts per switch. The main conclusion may be taken from the network size performance graphic that sums switch scalability up.

3.3.3 Environment

At first, several tests were run in a laptop with two Intel Core i7-4500U @ 1.80 Ghz and 2 GB of RAM @ 1600 MHz, using Mininet due to its fast deployment and customization as explored in section 2.2.1. Due to computer power limitations, the project was implemented under OpenStack Icehouse based on KVM, offering full virtualization with the support of a host OS.

During the realization of this work, the virtualization platform used by the ATNoG group[8], where this master was performed changed to Proxmox. This change did not require any change to the tests, and allowed for a greater plurality of execution environments. All controllers were installed according to instruction presented in Appendix A.

3.3.4 Framework deployment

The framework deployment used in performance tests is represented in Figure 3.1.

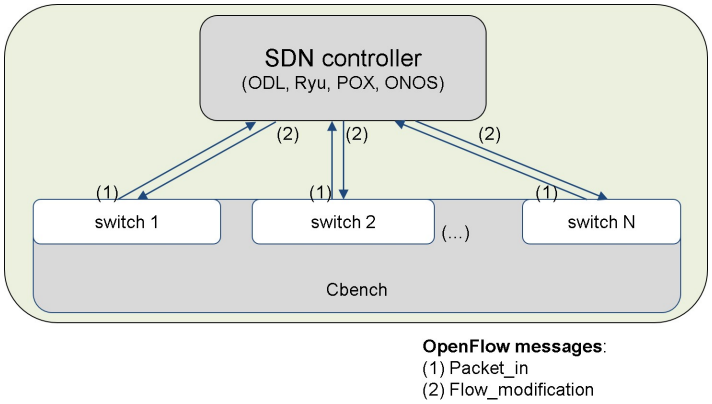


Figure 3.1: Schematic of flow exchange between both machines

These flow exchanges occur according to *Cbench*'s running mode as explained in sections 3.3.2.1 and 3.3.2.2.

As mentioned above, the framework presented was run in both OpenStack and Proxmox at different stages.

3.4 Enhancing mobile video transmission with VDSNet

Due to the continuous growth in the consumption and production of high quality video, the main motivation of VDSNet project is to define innovative solutions for Mobile User Generated Video (MUGV) services. The increasing use of services like Periscope[9] or Facebook

[8] https://atnog.av.it.pt/
[9] https://www.periscope.tv/

Live[10] to do live transmission from mobile phones will have a great impact in all networking industry. In order to face that, VDSNet covers the scenario of mobile video delivery through means of SDN technologies, giving the focus on uplink.

The comparison study carried out, both qualitatively and quantitatively, had the purpose of electing the SDN controller to be implemented in the VDSNet project.

3.4.1 VDSNet architecture description

This subsection presents the VDSNet architecture and its high level operation. The proposed design is presented in Figure 3.2.

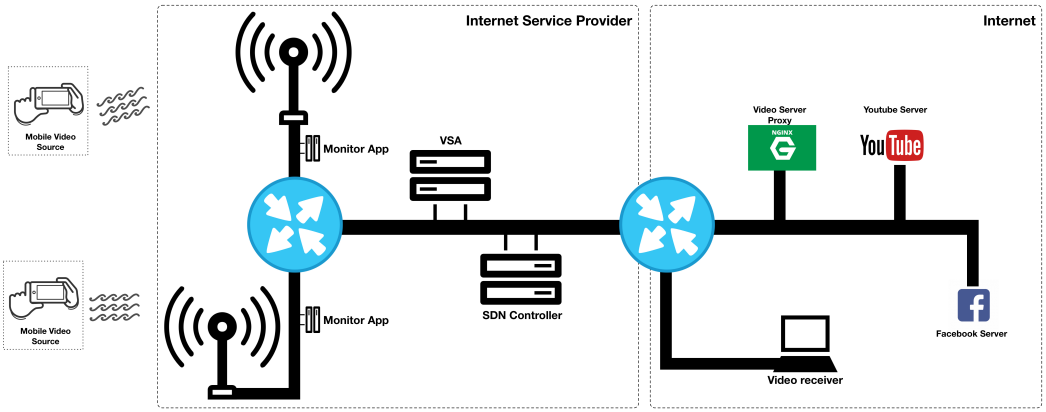


Figure 3.2: VDSNet architecture

There are two modules residing in the network, the Video SDN Application (VSA) and the SDN controller, while the Monitor App is located in the mobile terminal(s).

3.4.2 VDSNet Modules

This subsection provides a short description with the expected functionality of each module.

3.4.2.1 SDN Controller

By having a centralized view of the whole mobile network, the SDN controller is able to collect statistics from the network and therefore control routing paths according to the policies provided by VSA.

SDN controller is able to optimize the network considering the QoS policy control, the topology manager (by providing the view of the network path resources), the routing function and the flow management function. The comparison study carried out helped to determine which SDN controller was to be implemented in the VDSNet project.

[10]https://live.fb.com/

3.4.2.2 Video SDN Application (VSA)

VSA optimizes the network considering user and content-specific policies based on QoS policy control. VSA handles service requests by delivering to the SDN controller the requirements to the mobile video transmission in terms of QoS. In other words, the VSA receives information concerning the video upload and monitors the status of video flows by interacting with the SDN controller with QoS requests.

3.4.2.3 Monitor App

The Monitor App is a network monitoring application focused on detecting video flows in the network. It has access to all network flows that enter the ISP and is able to modify the VSA, sending the destination IP and port of the detected flow.

The Monitor App is composed by five different detection mechanisms, a whitelist that contains flows previously defined as video, a blacklist with flows that were previously defined as non-video, a network port analysis since some protocols always use the same port number, a hostname analysis to detect the existence of some keywords and a periodicity analysis to analyse the frequency of the incoming packets in order to compare them with previously stored packets.

3.5 Chapter considerations

This chapter addressed the chosen criteria taken into consideration in the qualitative comparison. Additionally it presented the setup requirements and the framework deployed overtime, as well as the benchmarking tool used and its working modes.

CHAPTER 4

SDN Controller Comparison

This chapter presents the analysis of each controller when considering the criteria described in the previous chapter.

Also, it presents latency and scalability results from running each SDN controller against *Cbench*. The performance tests carried out with *Cbench* in latency mode measure each controller latency in milliseconds. Otherwise, while running in throughput mode, it allows the introduction of different metrics to achieve several scalability results, such as switch and hosts scalability. The results were achieved by running both machines under OpenStack and Proxmox at different times.

The main results were published in two papers: "A Qualitative and Quantitative assessment of SDN Controllers" in International Young Engineers Forum on Electrical and Computer Engineering (YEF-ECE 2017) and "CREDENCE: A Carrier Grade Software Defined Networking Control Environment Based on the JAIN SLEE Component Model" in the 22nd IEEE Symposium on Computers and Communications (ISCC 2017).

4.1 Qualitative Comparison

Each controller was analysed individually, either by exploring them or based on official documentation provided by their creators. Based on the selected criteria, Table 4.1 was elaborated to sum up the qualitative comparison for each controller under test.

	POX	RYU	ODL	ONOS
NB interface	Yes	Yes	Yes	Yes
SB Interface	OF, OVSDB	OF, NETCONF, OVSDB, SNMP	OF, NETCONF, TL1, OVSDB, SNMP, BGS-LS	OF, OVSDB, NETCONF, SNMP, TL1
GUI	No	Yes (limited)	Yes (Web based)	Yes (Web based)
REST API	Yes	Yes	Yes	Yes
Programming Language	Python	Python	Java	Java
OpenFlow Support	OF v1.0	OF v1.0, 1.2, 1.3, 1.4, 1.5	OF v1.0,1.3,1.4	OF v1.0,1.3, 1.5
Modularity	No	No	Yes	Yes
Multi-threading Support	No	Yes	Yes	Yes
Documentation	Poor	Poor	Good	Good

Table 4.1: Qualitative Comparison

There are key factors for every project that may be taken into consideration when choosing a SDN controller. For small teaching or research projects POX and Ryu, besides having a lack of documentation for starters or having limitations regarding their REST API or supported SB protocols, have the advantage of needing much less computing capacity when compared to ODL or ONOS. Additionally, by being developed in Python, it facilitates the creation of simple (or more complex) applications and allows the controller to run anywhere.

Knowing that the most accepted/used OpenFlow versions are 1.0 and 1.3, POX becomes the only that lacks official support regarding version 1.3 and that may be a considerable factor when electing the desired SDN controller.

Despite being classified as being well documented, ODL lacks documentation in some areas, such as their YANG interface, mainly due to its constant evolution. A similar situation occurs with ONOS since there is a constant flow of new releases from both controllers. Still, both have a considerable amount of documentation in their sites and each time a controller is upgraded, documentation about that newer version is always released. Still, all of them, for being widely used, have countless small projects and use-cases that may be useful to everyone starting on SDNs.

Having multi-threading support is an advantageous criteria when considering big scenarios or real use-cases where the network is considerable big, but it can also be irrelevant in small teaching/researching purposes.

From all information provided by Table 4.1, POX can be negatively highlighted as it shows serious limitations regarding only supporting OpenFlow version 1.0 and the lack of a GUI for visualising SDNs in order to facilitate the network topology management. Additionally, POX shares with Ryu the lack of modularity support. The positive highlight goes to ODL and ONOS, both presenting a wide choice of SB protocols, a web based GUI and support to modularity and multi-threading.

Without electing a winner, we can clearly claim that ODL and ONOS are the most

featured controllers regarding our qualitative comparison.

4.2 Performance

In this section performance tests will be discussed by describing both latency and throughput results. From all input arguments referred before in section 2.3.1 about *Cbench*, the number of switches and hosts emulated per switch are the most changed arguments in order to obtain the switch scalability and hosts scalability.

The following subsections present the different tests run on each controller. Appendix B presents every needed feature for each controller under test.

The framework deployment was described in Figure 3.1 with the setup requirements presented in Table 3.1.

4.2.1 Controller Latency

We tested each controller for latency results by running *Cbench* in latency mode as described in section 3.3.2.1. To achieve the results in Table 4.2, one switch was emulated, with a fixed number of hosts (10^5) for at least 10 seconds, repeating each test 10 times for each controller.

```
cbench -c <controller ip> -p 6633 -s 1 -m 10000 -l 10 -M 1000000
```

After this process, the following latency values were obtained:

Controller	Latency (ms)
POX	1.48 ± 0.015
Ryu	0.23 ± 0.002
ODL	0.34 ± 0.02
ONOS	0.63 ± 0.044

Table 4.2: Latency (ms)

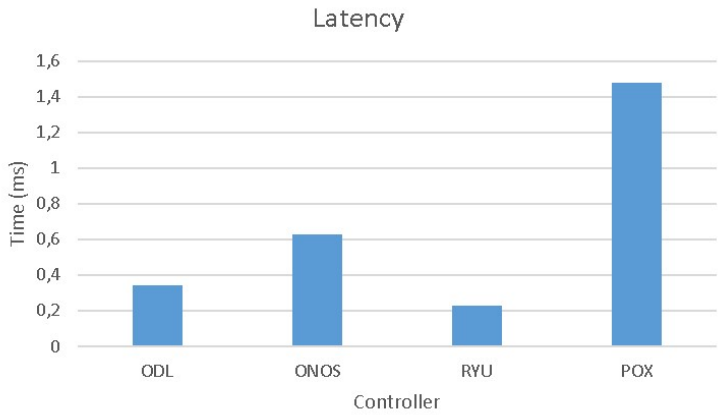


Figure 4.1: Latency graph

Analysing these latency tests, Ryu had the lowest latency with 0.23 milliseconds, followed up closely by ODL. ONOS achieved 0.63 milliseconds while POX presented the worst latency result with 1.48 milliseconds.

Considering these latency results achieved, it was decided to study each controller switch scalability in latency mode, by increasing the number of emulated switches until 128. The number of hosts per switch was kept in (10^5) and each test was repeated 10 times for at least 5 seconds. The following commands were run and results achieved are represented in Figure 4.2.

```
cbench -c <controller ip> -p 6633 -s 1 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 2 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 4 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 8 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 16 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 32 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 64 -m 5000 -l 10 -M 1000000
cbench -c <controller ip> -p 6633 -s 128 -m 5000 -l 10 -M 1000000
```

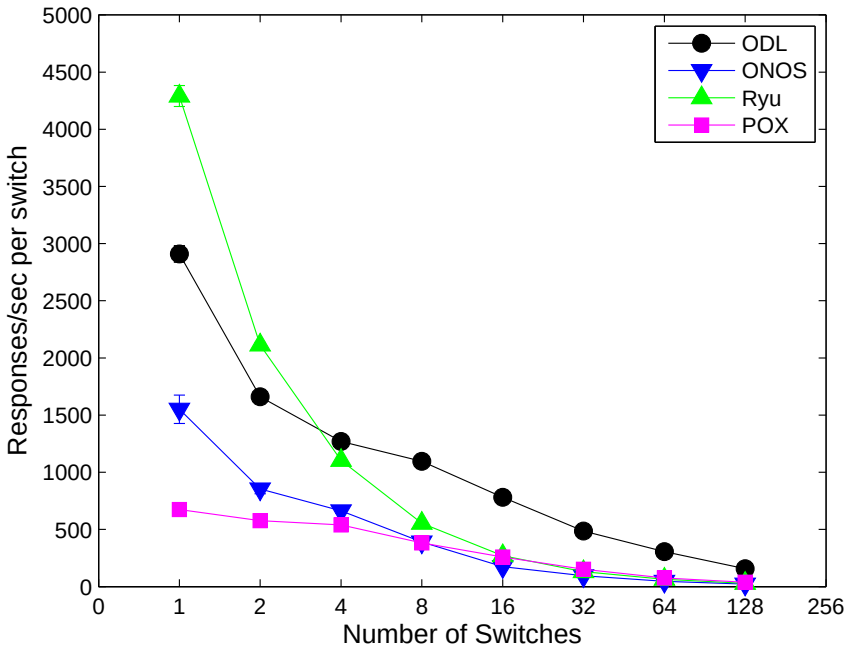


Figure 4.2: Switch scalability in latency mode

Number of Switches	ODL	ONOS	RYU	POX
1	2910 ± 7,14	1551 ± 123,96	4291 ± 91,73	674 ± 4,24
2	1661 ± 21,69	854,5 ± 41,85	2112,5 ± 23,45	445 ± 35,50
4	1269 ± 16,04	662,25 ± 22,78	1103,75 ± 4,73	539,5 ± 61,69
8	1095,62 ± 10,82	390,75 ± 13,01	554 ± 3,07	381,38 ± 13,12
16	780,44 ± 6,10	174,75 ± 3,05	270,81 ± 1,96	258,63 ± 2,48
32	485,19 ± 2,52	95,0625 ± 1,65	130,59 ± 0,42	151,47 ± 1,03
64	305,2 ± 1,96	45,5625 ± 0,46	64,14 ± 0,2	75,48 ± 0,14
128	157,04 ± 0,22	22,59375 ± 0,26	27,69 ± 0,06	37.53 ± 0,12

Table 4.3: Latency switch scalability (responses/sec)

By analysing Figure 4.2 and into detail in Table 4.3, despite Ryu's amazing performance for 1 and 2 emulated switches, ODL overcomes when considering 4 or more by doubling any controller's amount of responses. Furthermore, Ryu quickly decreases its performance reaching for less than 500 responses per second.

After migrating from OpenStack to Proxmox, some new latency results were achieved considering SDN controllers' versions previously tested to compare both environments.

Considering the newest results achieved when comparing the same controllers' versions in both environments, results were very close to each other. Results are presented in Figure 4.3, showing how similar results were in both environments.

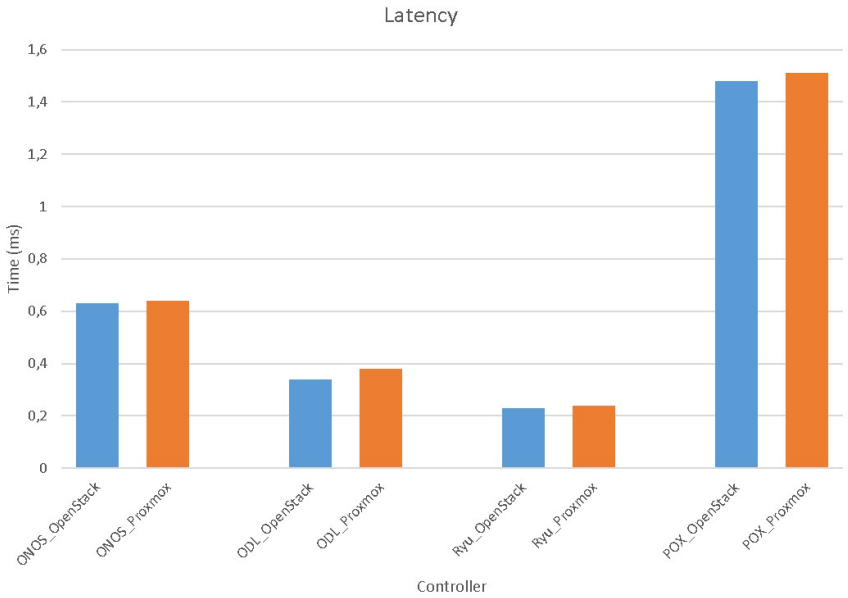


Figure 4.3: Latency environments comparison

A newer version of ODL was installed and new latency results were achieved as shown in Figure 4.4. When comparing ODL Carbon, release on 26 May 2017 with Beryllium (SR3) the

amount of flows decreases from approximately 2900 to 2600. This difference means a 0,04 ms increase in the latency value (0.38 milliseconds and 0.34 milliseconds, respectively). The comparison with the new version of ODL being considered is presented in Figure 4.4.

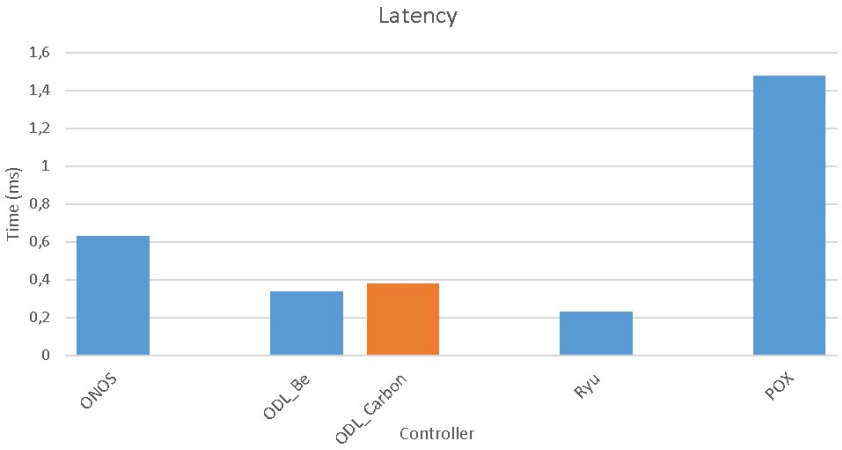


Figure 4.4: Latency graph with ODL comparison

Considering the initial confidence interval of ODL, the newest result achieved did not change ODL's latency performance, keeping the second lowest latency value among all controllers under test.

4.2.2 Controller Throughput

By running *Cbench* in throughput mode, several results were achieved, as described in the following sections. Starting with the switch scalability results, the number of switches considered varied from 8 to 128 in order to check each controller behaviour with the increasing number of switches. Thereafter, the number of hosts emulated per switch was increased from 1000 to 10 million with a fixed number of switches.

4.2.2.1 Switch scalability

This subsection covers different switch scalability tests by varying the number of switches emulated while the number of hosts emulated per switch was 100000. Each test was run for at least 10 seconds and the following graphics show the average of flows on each switch considering 10 different tests. Before each graphic, the command used to run *Cbench* with distinct parameters is presented.

```
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 1000000 -t
```

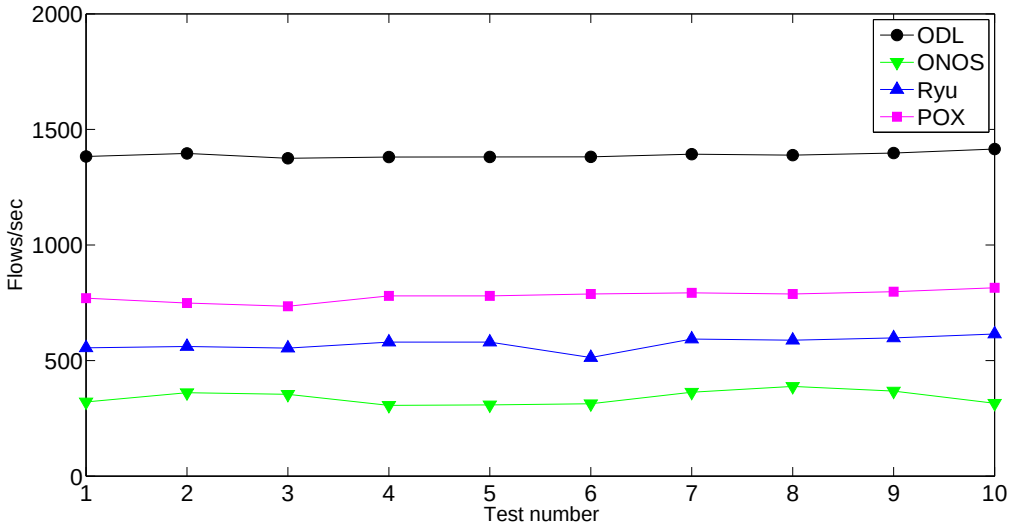


Figure 4.5: Evaluation throughput test for 8 switches

When considering 8 switches, every controller shows a constant value due to the low power required to emulate 8 switches with 100000 hosts each.

ODL counted around 1400 flows per second, followed up by POX with 800 and RYU a bit below 600 flows per second. ONOS presented the lowest number of flows, between 300 and 350 with a little variation.

```
cbench -c <controller ip> -p 6633 -s 16 -m 100000 -l 10 -M 1000000 -t
```

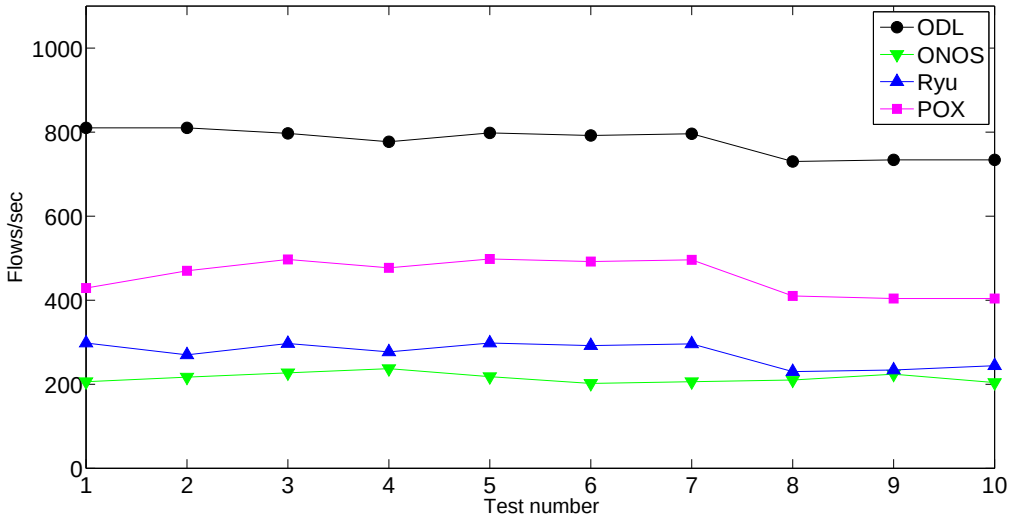


Figure 4.6: Evaluation throughput test for 16 Switches

With 16 switches emulated, each controller presented a similar behaviour when compared

to the previous graphic with just 8 switches.

```
cbench -c <controller ip> -p 6633 -s 32 -m 100000 -l 10 -M 1000000 -t
```

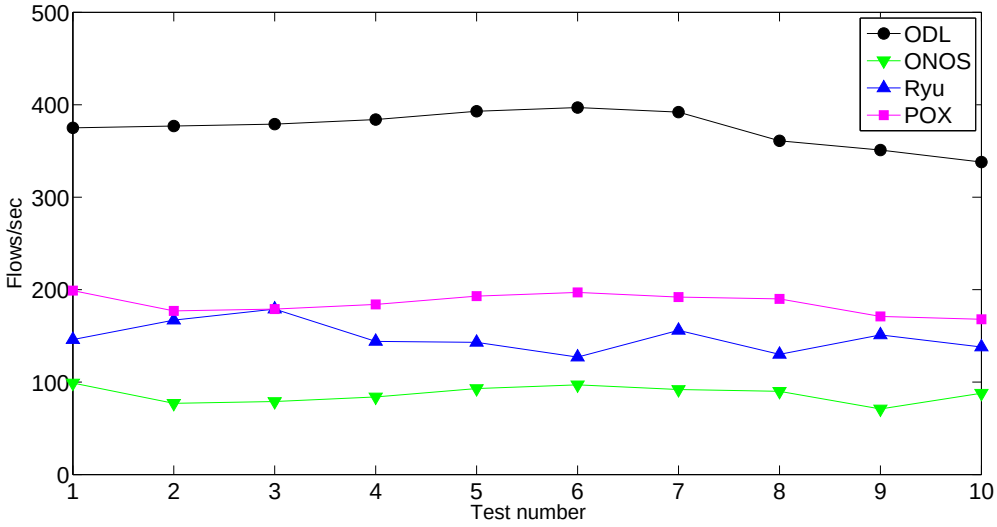


Figure 4.7: Evaluation throughput test for 32 Switches

Ryu started to show some inconsistency on the average of the 10 tests considered.

```
cbench -c <controller ip> -p 6633 -s 64 -m 100000 -l 10 -M 1000000 -t
```

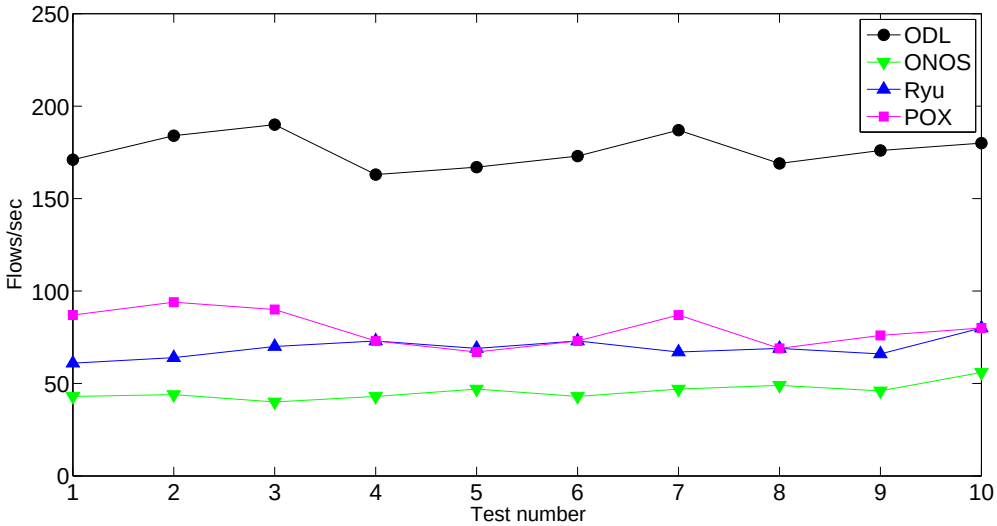


Figure 4.8: Evaluation throughput test for 64 Switches

When achieving a number of switches higher than 32, every controller starts to present a significant variation due to the computer power required. When considering 64 switches, ODL presented values varying from 160 to 200 flows per second while POX presented a similar swing achieving values between 60 and 100 flows per second, being surpassed by Ryu in some tests. Ryu and ONOS kept presenting consistent values.

```
cbench -c <controller ip> -p 6633 -s 128 -m 100000 -l 10 -M 1000000 -t
```

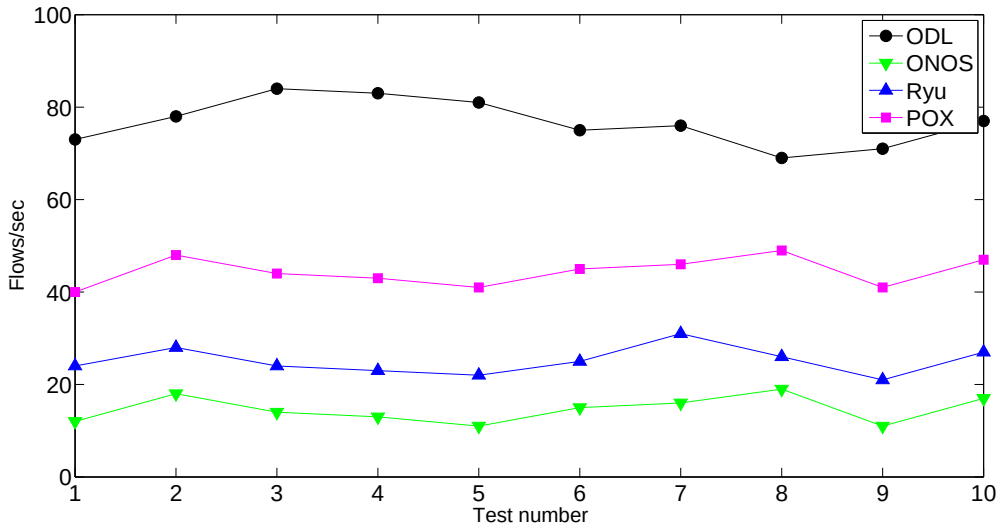


Figure 4.9: Evaluation throughput test for 128 Switches

For 128 emulated switches, every controller keeps presenting a variation due to the demanded computer power from the controller but also showing a well defined range of values. ODL presents between 70 and 85 flows per second. POX varies between 40 and 50, followed up by Ryu with 20 to 30 flows and ONOS with a number of flows from 10 to 20 per second.

In Appendix C the same results under detail in Table C.1 are presented.

Considering these switch scalability results, by extending them from 1 to 512 emulated switches, graphic 4.10 presents the network size performance, summarizing obtained results regarding switch scalability:

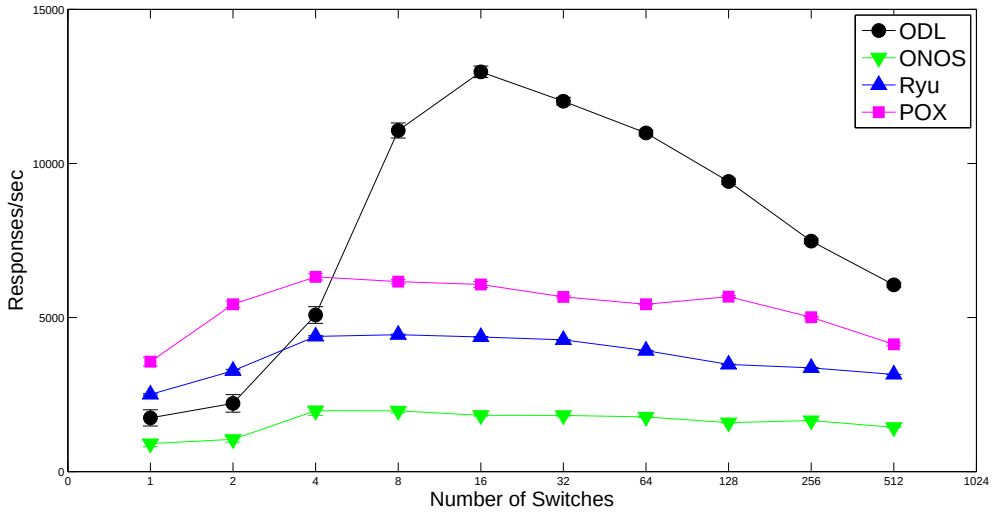


Figure 4.10: Network Size Performance

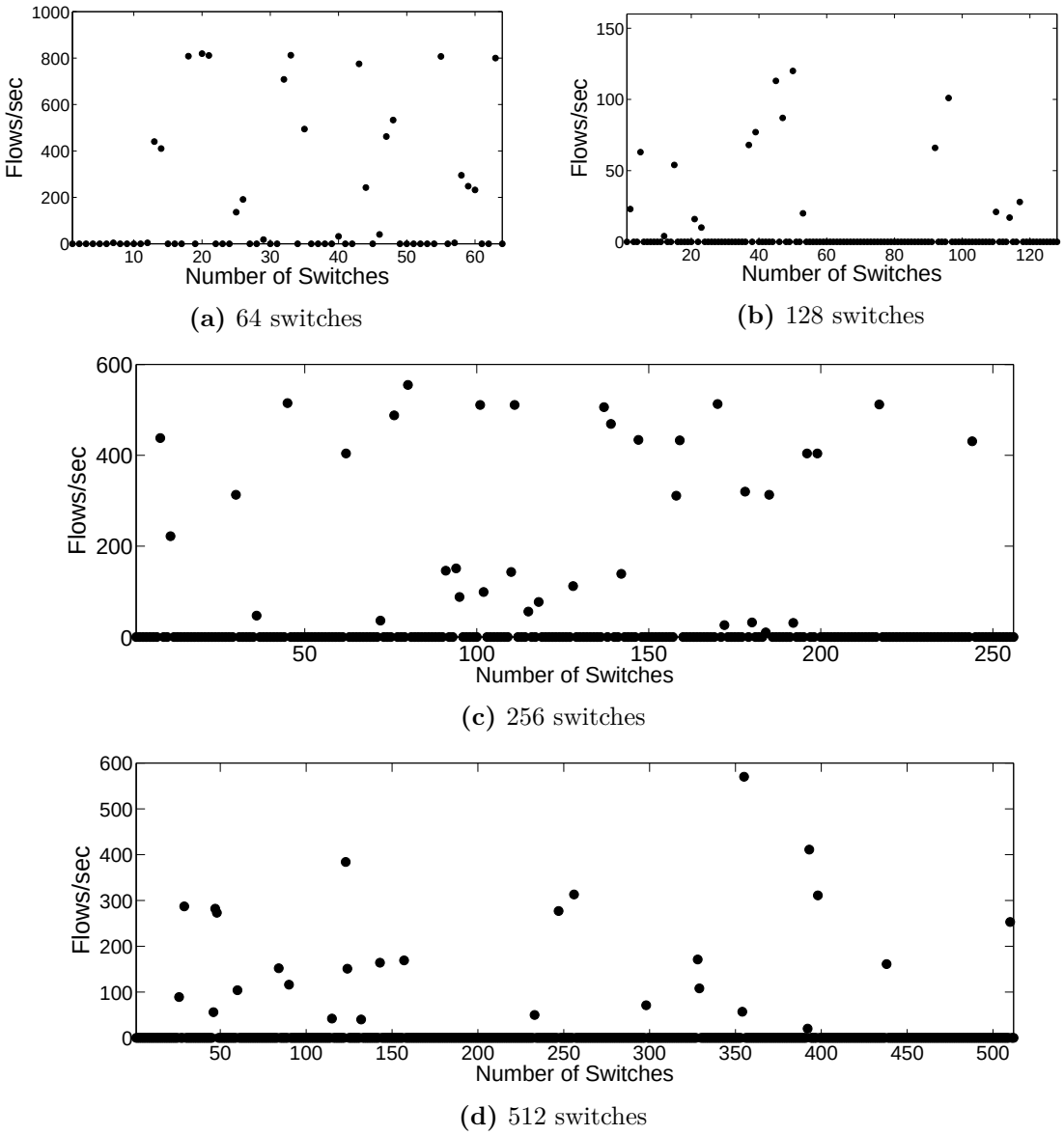
Regarding switch scalability throughput tests (Figure 4.10), ODL clearly achieves better rates than any other controller under test, achieving above 10000 responses per second for 8, 16 and 32 switches. POX places second with around 6000 responses per sec while Ryu and ONOS got close to 4000 and 2000 responses per sec, respectively. This network size performance graphic summarizes each SDN controller behaviour with the increasing number of switches, supporting the election of ODL as the best SDN controller in this regard.

4.2.2.2 Tests Consistency

While running these tests, in every controller, some tests returned a considerable amount of zero responses (failed tests), when considering 32 or more switches. This reflected the increasing demand for processing effort required with the increasing number of switches, on behalf of the controller. POX and Ryu presented a low number of failed tests, but ODL and ONOS showed more than 50% of failed tests for 256 and 512 emulated switches. Those failed tests are shown in Table 4.4, whose percentage increases with the number of switches. Due to their highest complexity and therefore computer power required, ODL and ONOS present an high value of failed tests, specially when considering above 128 switches, considering the resources used for the experimental evaluation.

Controller	Number of Switches	Number of Tests	Failed Tests (%)
ONOS	32	320	37
	64	640	61
	128	1280	75
	256	2560	85
	512	5120	92
ODL	32	320	8
	64	640	14
	128	1280	42
	256	2560	61
	512	5120	88

Table 4.4: Failed Tests for ONOS and ODL



(a) 64 switches

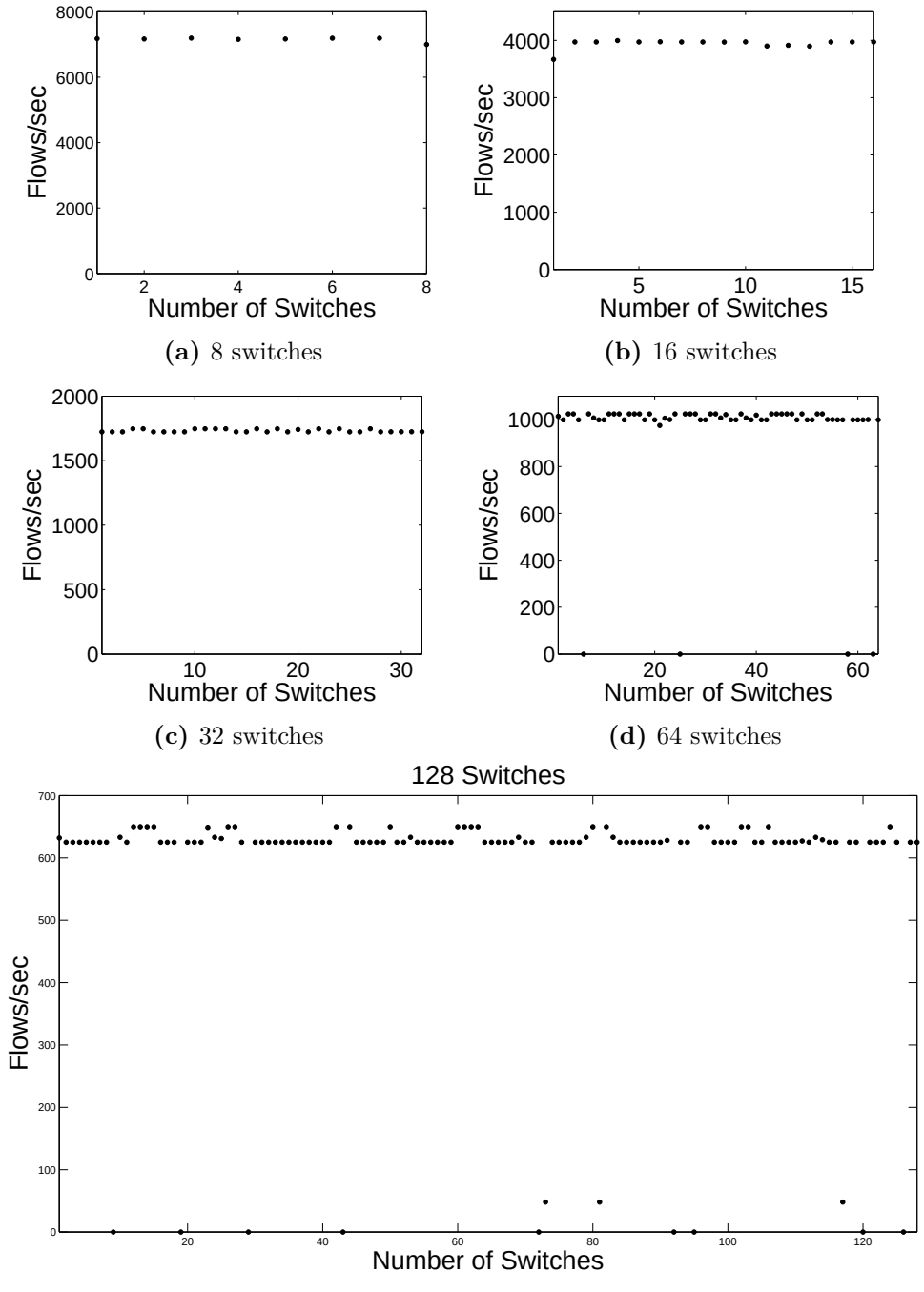
(b) 128 switches

(c) 256 switches

(d) 512 switches

Figure 4.11: ONOS switch distribution

Since ONOS presents the highest amount of failed tests, Figure 4.11 shows the distribution of flows through the number of switches emulated. On the contrary, POX and RYU showed better consistency even when considering an higher number of switches. The distribution of flows per switch for POX is presented in Figure 4.12.



(a) 8 switches

(b) 16 switches

(c) 32 switches

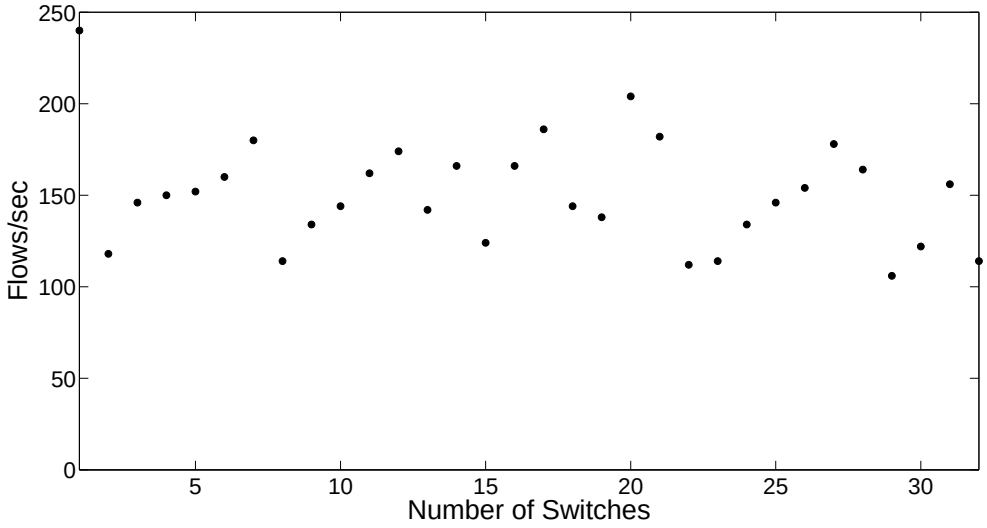
(d) 64 switches

(e) 128 switches

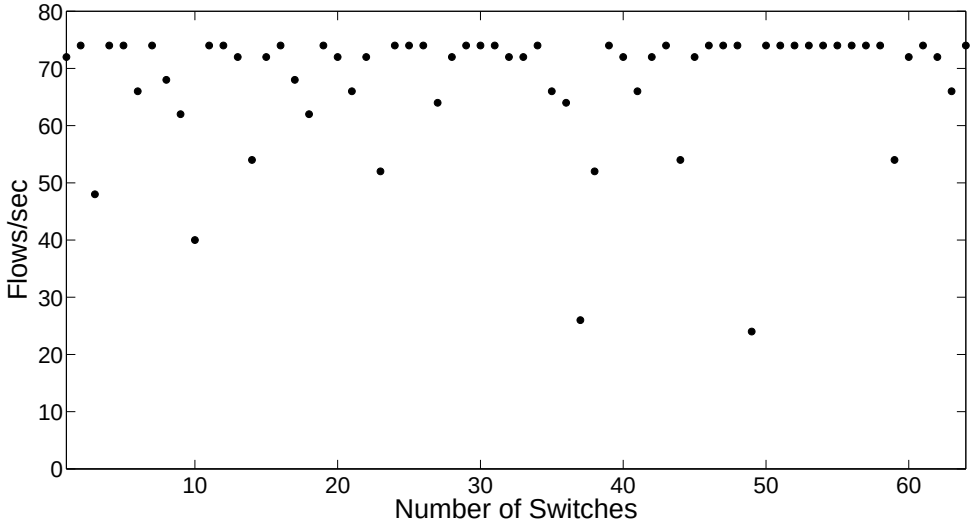
Figure 4.12: POX switch distribution

POX showed zero failed tests for 8, 16 and 32 emulated switches and had just four failed tests when considering 64 switches, approximately 6%. The most demanding test, with

128 emulated switches, POX had less than 10% failed tests, considered a low number when compared to 75% from ONOS.



(a) 32 switches



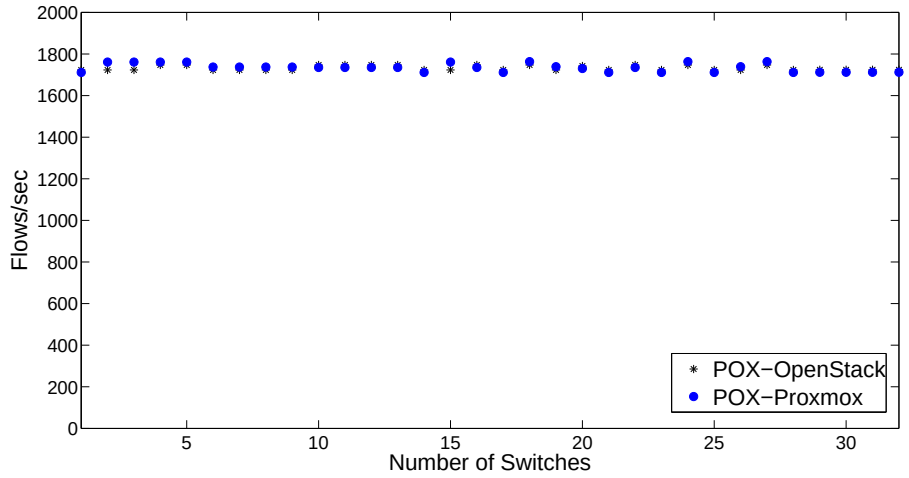
(b) 64 switches

Figure 4.13: Ryu switch distribution

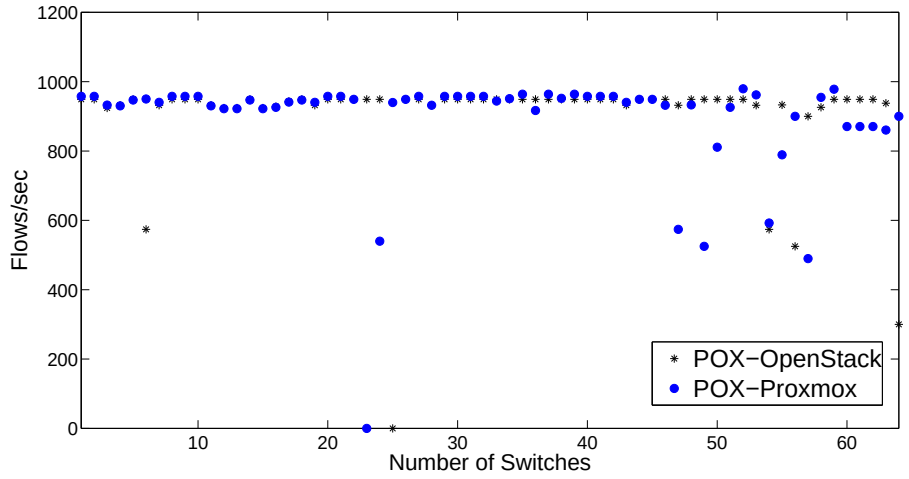
POX showed a great consistency on the number of flows and analysed failed tests were represented with zero flows in a specific switch. Unlike POX, Ryu showed in Figure 4.13 a bigger dispersion around the average value. Considering 32 switches (Figure 4.13a), with an average of, approximately, 150 flows per second but values distributed evenly between the 100 and 200 marks. Despite not presenting failed tests with zero flows in any specific switch, Ryu achieved a reduced number of flows in several switches in Figure 4.13b.

When running *Cbench* in throughput mode, either in OpenStack or Proxmox, obtained results were solid. That led to the conclusion that despite choosing the environment, that

choice will not affect the performance of any controller. As an example, Figure 4.14 shows the comparison between environments for POX.



(a) 32 switches



(b) 64 switches

Figure 4.14: POX Environment Comparison

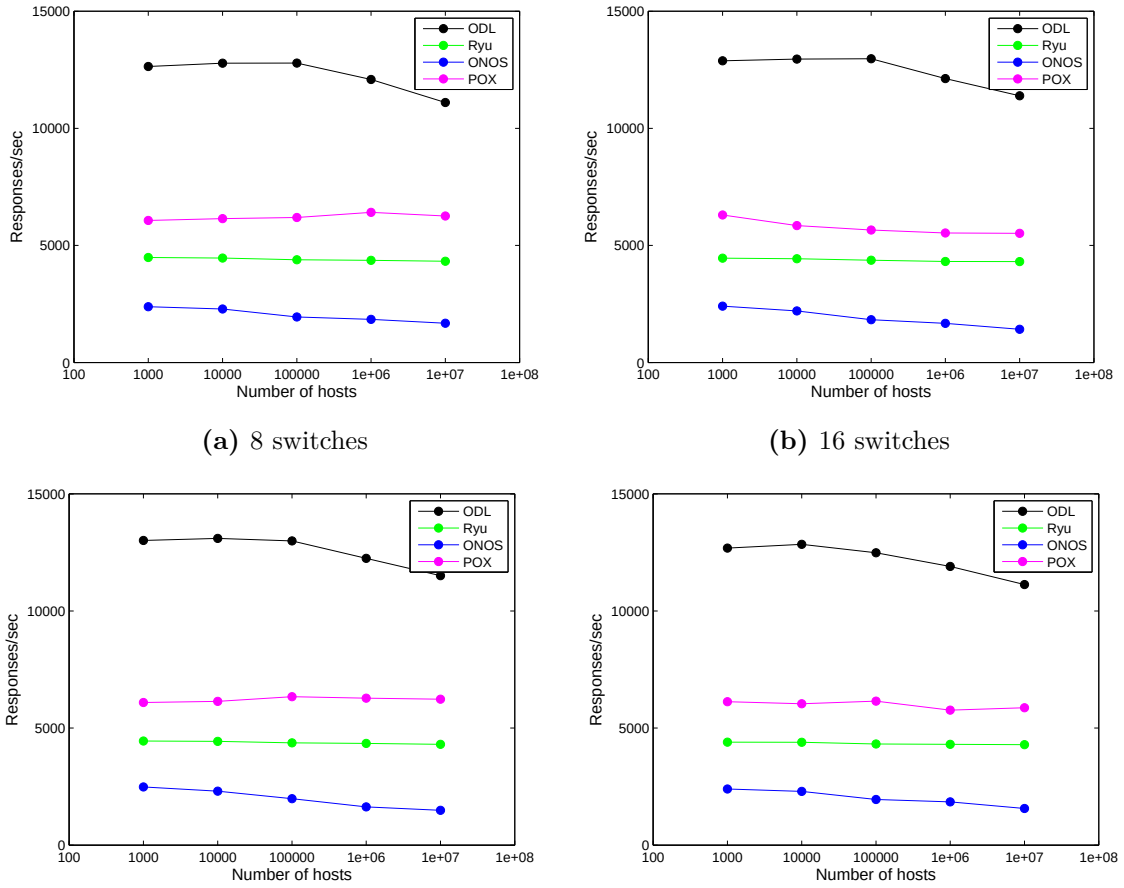
When considering 32 switches (Figure 4.14a), obtained results from both environments were solid, varying from 1700 to 1800 flows per second. Also, when considering 64 switches (Figure 4.14b), obtained results varied between 900 and 1000 flows per second with a few exceptions, since some switches showed around 600 flows. Presented exceptions in Figure 4.14b occurred independently of the environment, since the amount of switches presenting a lower amount of flows for each environment is similar.

4.2.2.3 Hosts scalability

To verify the impact of the number of hosts emulated per switch, we also ran *Cbench* in throughput mode but by having a fixed number of switches (8, 16, 32 and 64) and varying

the number of hosts per switch from 1000 to 10 million. The following commands were used to emulate 8 switches:

```
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 1000 -t
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 10000 -t
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 100000 -t
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 1000000 -t
cbench -c <controller ip> -p 6633 -s 8 -m 100000 -l 10 -M 10000000 -t
```



(a) 8 switches

(b) 16 switches

(c) 32 switches

(d) 64 switches

Figure 4.15: Increasing number of hosts emulated per switch

The impact of increasing the number of hosts emulated per switch is very low for both Ryu and POX, despite the number of switches considered. That same impact is noticeable for ODL and ONOS, mainly when considering 100.000 or more hosts, according to Figure 4.15.

Analysing into detail Figures 4.15a and 4.15b, POX and Ryu did not show any irregular behaviour with the increasing of hosts emulated per switch, achieving, approximately, 6000 and 4400 responses per sec, respectively. ONOS and mainly ODL presented a small drop when considering 1 million or more hosts. ODL started around 13000 responses per second for 1000 hosts and was able to keep that performance until it decayed to around 12000 and 110000 responses for 1 and 10 million hosts, respectively. ONOS starts with 2400 responses

for 1000 hosts and decreases its performance until 1400 responses when considering 10 million hosts.

This different behaviour between both Python-based controllers (POX and Ryu) and both Java-based controllers (ODL and ONOS) is related to their complexity and so computer power required when considering an higher number of hosts to emulated per switch apart from the number of switches, as demonstrated in Figure 4.15.

4.3 VDSNet Project integration

Within the scope of the project, from the characteristics summarized in Table 4.1, ODL and ONOS were the most-featured SDN controllers.

Regarding latency, Ryu had the best results, followed up closely by ODL. Although it is worth noticing that when considering more than 4 emulated switches, ODL achieved better latency results tan Ryu. While in throughput tests, switch and host scalability, ODL achieved the highest number of flows per second by far. Considering all the performance tests achieved, ODL achieved the best performance in general.

In conclusion, ODL was implemented in VDSNet Project.

4.4 Chapter considerations

This chapter addressed the characteristics of the controllers following the selected criteria. Also, it presented a latency outcome, as well as, switch and host scalability results. OpenStack and Proxmox environments were also compared.

CHAPTER 5

Conclusion

With the aim to improve 5G network research, the comparative study carried out explores the most recent versions of famous SDN controllers. Also, it compares them in terms of performance, mainly switch and hosts scalability.

In chapter 2 an exhaustive exploration of the SDN architecture shows some of the SDN advantages like more control, adaptability and overall cost reduction. This cost reduction is reflected in both CAPEX and OPEX since SDN gives the possibility of modelling a physical networking environment into software. Additionally, unlike traditional networks, SDN allows a significant reduction on time spent to substitute or reconfigure the equipment by using a central management tool, facilitating any network equipment update. The ability for vendors to extend the capabilities through management APIs in order to develop applications according to their needs is also highly valuable. Several SDN controllers are explored in this same chapter, as well as, additional SDN tools and environments related to the use of *Cbench*. *Cbench* is a benchmarking tool often used to run performance tests on SDN controllers either in latency or throughput mode.

Due to the amount of SDN controllers developed, in chapter 3 four SDN controller were elected to perform a qualitative and quantitative comparison, as well as, several criteria to compare their characteristics. The setup requirements and framework deployment are presented. Also, the VDSNet project is presented along with its architecture and modules.

In chapter 4 is presented a qualitative table (Table 4.1) that summarizes each SDN controller characteristics. Additionally are presented results regarding both latency and performance tests. Within performance tests, the main focus was in switch and hosts scalability. The main conclusions and results regarding performance tests are also described.

To conclude, this dissertation contributed towards SDN controllers effort by studying the most recent versions of each controller. This comparison results were validated and accepted as a paper to International Young Engineers Forum on Electrical and Computer Engineering (YEF-ECE) and The 22nd IEEE Symposium on Computers and Communications (ISCC 2017).

5.1 Future Work

As it was previously mentioned, there are several SDN controllers under development with constant updates being released. Therefore, by upgrading to their latest version, new results can be achieved, resulting into new comparisons. Additionally, by using more powerful machines, better results can also be achieved, specially when considering SDN controllers that support multi-threading.

References

[1] A. Ravanshid, P. Rost, D. S. Michalopoulos, V. V. Phan, H. Bakker, D. Aziz, S. Tayade, H. D. Schotten, S. Wong, and O. Holland, "Multi-connectivity functional architectures in 5g", in *Communications Workshops (ICC), 2016 IEEE International Conference on*, IEEE, 2016, pp. 187–192.

[2] W. H. Chin, Z. Fan, and R. Haines, "Emerging technologies and research challenges for 5g wireless networks", *IEEE Wireless Communications*, vol. 21, no. 2, pp. 106–112, 2014.

[3] *The zettabyte era — trends and analysis – cisco*, `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`, Accessed: 2017-04-30.

[4] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the impact of video quality on user engagement", *Communications of the ACM*, vol. 56, no. 3, pp. 91–99, 2013.

[5] O. Awobuluyi, J. Nightingale, Q. Wang, and J. M. Alcaraz-Calero, "Video quality in 5g networks: Context-aware qoe management in the sdn control plane", in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*, IEEE, 2015, pp. 1657–1662.

[6] *Cisco visual networking index: Global mobile data traffic forecast update*, `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`, Accessed: 2017-04-30.

[7] *Telco spending on sdn and nfv forecast to reach us$157 billion by 2020*, `http://www.telecomtv.com/articles/sdn/telco-spending-on-sdn-and-nfv-forecast-to-reach-us-157-billion-by-2020-12859/`, Accessed: 2017-05-10.

[8] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks.", *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[9] O. N. Foundation. (2016). SDN Architecture. Accessed: 2017-05-30, [Online]. Available: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-521_SDN_Architecture_issue_1.1.pdf`.

[10] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[11] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "The evolution of sdn and openflow: A standards perspective", *IEEE Computer Society*, vol. 47, no. 11, pp. 22–29, 2014.

[12] Á. L. V. Caraguay, A. B. Peral, L. I. B. López, and L. J. G. Villalba, "Sdn: Evolution and opportunities in the development iot applications", *International Journal of Distributed Sensor Networks*, 2014.

[13] *Building blocks of sdn network*, `https://nutanshinde.wordpress.com/category/sdn/`, Accessed: 2017-05-30.

[14] Y. I. Daradkeh, M. ALdhaifallah, D. Namiot, and M. Sneps-Sneppe, "On standards for application level interfaces in sdn",

[15] B. N. Astuto, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking : Past , Present , and Future of Programmable Networks", *IEEE Communications Surveys & Tutorials*, vol. 16, pp. 1617–1634, 2014.

[16] F. Meneses, D. Corujo, C. Guimaraes, and R. L. Aguiar, "Extending sdn to end nodes towards heterogeneous wireless mobility", in *Globecom Workshops (GC Wkshps), 2015 IEEE*, IEEE, 2015, pp. 1–6.

[17] ——, "Multiple flow in extended sdn wireless mobility", in *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, IEEE, 2015, pp. 1–6.

[18] *Odl beryllium (be) - the fourth release of opendaylight*, `https://www.opendaylight.org/odlbe`, Accessed: 2017-05-15.

[19] *Technical white paper*, `http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf`, Accessed: 2017-05-15.

[20] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*. " O'Reilly Media, Inc.", 2013.

[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[22] D. Erickson, "The beacon openflow controller", *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, p. 13, 2013. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2491185.2491189`.

[23] Z. Cai, A. Cox, and E. T. S. Ng, "Maestro: A System for Scalable OpenFlow Control", *Cs.Rice.Edu*, p. 10, 2011.

[24] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight sdn controller", in *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, IEEE, 2014, pp. 671–676.

[25] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of sdn controllers", in *Communications (ICC), 2013 IEEE International Conference on*, IEEE, 2013, pp. 3504–3508.

[26] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible openflow-controller benchmark", in *Software Defined Networking (EWSDN), 2012 European Workshop on*, IEEE, 2012, pp. 48–53.

[27] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks.", *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[28] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based comparison and selection of software defined networking (sdn) controllers", in *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, IEEE, 2014, pp. 1–7.

[29] Y.-D. Lin and C.-H. Chang, "OpenFlow Version Roadmap", pp. 1–15, 2015.

[30] B. Pfaff and B. Davie, "The open vswitch database management protocol", 2013.

[31] R. Enns, "Network Configuration Protocol (NETCONF)", pp. 6–9, 2011.

[32] F. Keti and S. Askar, "Emulation of software defined networks using mininet in different simulation environments", in *Intelligent Systems, Modelling and Simulation (ISMS), 2015 6th International Conference on*, IEEE, 2015, pp. 205–210.

[33] R. Kumar, N. Gupta, S. Charu, K. Jain, and S. K. Jangir, "Open source solution for cloud computing platform using openstack", *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 5, pp. 89–98, 2014.

[34] D. Freet, R. Agrawal, J. J. Walker, and Y. Badr, "Open source cloud management platforms and hypervisor technologies: A review and comparison", in *SoutheastCon, 2016*, IEEE, 2016, pp. 1–8.

[35] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of sdn/openflow controllers", in *Proceedings of the 9th central & eastern european software engineering conference in russia*, ACM, 2013, p. 1.

[36] *Opendaylight user guide*, `https://www.opendaylight.org/sites/opendaylight/files/bk-user-guide.pdf`, Accessed: 2017-05-01.

[37] *Yang user interface (yangui) in opendaylight*, `http://events.linuxfoundation.org/sites/events/files/slides/YANGUI-metz-malachovsky-sebin-ODL-Summit-final-July29.pdf`, Accessed: 2017-05-30.

APPENDIX A

Installation Guide

This appendix presents installation instructions for each controller.

A.1 ODL

ODL requires Oracle Java 1.7 for Beryllium (SR3) or 1.8 for more recent versions:

```
sudo apt-get install oracle-java7-installer oracle-java7-set-default -y
or
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
```

ODL also requires Karaf and Maven:

```
wget http://archive.apache.org/dist/karaf/3.0.5/apache-karaf-3.0.5.tar.gz
wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
tar -zxvf apache-karaf-3.0.5.tar.gz
tar -zxvf apache-maven-3.3.9-bin.tar.gz
```

Then, download the OpenDaylight's karaf distribution *.tar* package.

```
wget https://github.com/opendaylight/controller/archive/release/beryllium-sr3.tar.gz
tar -xvf beryllium-sr3.tar.gz
```

Setting JAVA_Home environment variable by adding the following line to the *bashrc* file:

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle
```

Then run the file:

```
source ~/.bashrc
```

Run OpenDaylight:

```
cd beryllium-sr3
./bin/karaf
```

A.2 ONOS

ONOS requires Oracle Java 1.8:

```
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
```

ONOS also requires Karaf and Maven:

```
wget http://archive.apache.org/dist/karaf/3.0.5/apache-karaf-3.0.5.tar.gz
wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
tar -zxvf apache-karaf-3.0.5.tar.gz
tar -zxvf apache-maven-3.3.9-bin.tar.gz
```

Then, a copy from ONOS repository needs to be downloaded using git command:

```
git clone https://gerrit.onosproject.org/onos
```

Setting JAVA_Home environment variable:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

Export ONOS_ROOT environment variable in the shell profile:

```
export ONOS_ROOT=~/onos
source ONOS_ROOT/tools/dev/bash_profile
```

Build ONOS:

```
cd ~/onos
mvn clean install
```

A.3 Ryu

Ryu requires some Python dependencies and can be downloaded from the official repository:

```
sudo apt-get install python-pip python-dev
git clone git://github.com/osrg/ryu.git
cd ryu
```

A.4 POX

POX requires Python 2.7. In order to install the latest branch from POX repository (github):

```
git clone http://github.com/noxrepo/pox
cd pox
```

APPENDIX B

Features Guide

This appendix presents each controller needed feature in order to obtain presented results at chapter 4.

B.1 ODL

ODL needs, at least, the following features to be installed after launching the controller:

```
opendaylight-user@root>feature:install odl-l2switch-switch
opendaylight-user@root>feature:install odl-l2switch-all
opendaylight-user@root>feature:install odl-openflowplugin-flow-services
opendaylight-user@root>feature:install odl-openflowplugin-drop-test
opendaylight-user@root>dropallpacketsrpc on
```

There are several additional features that may be needed to run specific tests or to install interfaces, such as DLUX.

```
opendaylight-user@root>feature:install odl-dlux-all
```

These features are available in ODL version Beryllium (SR3). Latest versions may change the name of some features or even include them in different packages.

B.2 ONOS

On the other hand, the following APPS need to be exported before launching the controller:

```
export ONOS_APPS=drivers,openflow,proxyarp,mobility,fwd
```

B.3 Ryu

Unlike ODL or ONOS, while launching Ryu, the following app was launched at the same time:

```
ryu/app/simple_switch.py
```

B.4 POX

Last, to start POX:

```
./pox.py  forwarding.l2_learning
```

APPENDIX C

Table with throughput results

C.1 Detailed throughput results

The following table presents detailed results used to achieve graphics presented in section 4.2.2.1, from 8 to 128 emulated switches.

	POX					Ryu					ONOS					ODL				
Test Number	8	16	32	64	128	8	16	32	64	128	8	16	32	64	128	8	16	32	64	128
1	770	429	199	87	40	555	298	146	61	24	321	206	99	43	12	1383	810	375	171	73
2	749	470	177	94	48	561	270	167	64	28	361	217	77	44	18	1396	810	377	184	78
3	735	497	179	90	44	554	297	179	70	24	354	227	79	40	14	1375	797	379	190	84
4	780	477	184	73	43	580	277	144	73	23	306	237	84	43	13	1381	777	384	163	83
5	780	498	193	67	41	580	298	143	69	22	308	218	93	47	11	1381	798	393	167	81
6	788	492	197	73	45	513	292	127	73	25	313	202	97	43	15	1381	792	397	173	75
7	793	496	192	87	46	593	296	156	67	31	363	206	92	47	16	1393	796	392	187	76
8	788	410	190	69	49	588	230	130	69	26	388	210	90	49	19	1389	730	361	169	69
9	798	404	171	76	41	598	234	151	66	21	368	224	71	46	11	1398	734	351	176	71
10	815	404	168	80	47	615	244	138	80	27	315	204	88	56	17	1415	734	338	180	77

Table C.1: Detailed table - throughput results